Hadoop

云计算一体机

实践指南

李宁　王东亮　等编著



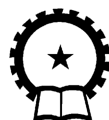
机械工业出版社
CHINA MACHINE PRESS

VISIT…

LANZAROTE Caliente.com

Hadoop 云计算一体机实践指南

李宁　王东亮　等编著



机 械 工 业 出 版 社

全书分为3篇：第1篇(理论部分)对云计算、Hadoop及Linux操作系统进行了简单介绍；第2篇(基础实践部分)主要详细介绍了CentOS系统的安装和集群的搭建、Hadoop集群的常用命令及管理应用等；第3篇(项目实训部分)主要以实际项目开发为例，从易到难，对源程序进行了详细解释。

本书以详细的实践操作介绍为特色，可作为电子、通信、自动化、计算机等电类专业Hadoop云计算教学系统课程实验教材，也可供Hadoop系统相关工程技术人员参考。

图书在版编目(CIP)数据

Hadoop云计算一体机实践指南/李宁等编著. —北京：机械工业出版社，2013.8

ISBN 978-7-111-43496-2

Ⅰ.①H… Ⅱ.①李… Ⅲ.①数据处理软件 Ⅳ.①TP274

中国版本图书馆CIP数据核字(2013)第175911号

机械工业出版社(北京市百万庄大街22号 邮政编码100037)

策划编辑：顾 谦 责任编辑：顾 谦

版式设计：常天培 责任校对：陈立辉

封面设计：赵颖喆 责任印制：乔 宇

北京铭成印刷有限公司印刷

2013年9月第1版第1次印刷

184mm×260mm·13.75印张·337千字

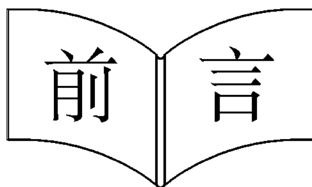
0001—3000册

标准书号：ISBN 978-7-111-43496-2

定价：39.90元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：(010)88361066	教材网：http://www.cmpedu.com
销售一部：(010)68326294	机工官网：http://www.cmpbook.com
销售二部：(010)88379649	机工官博：http://weibo.com/cmp1952
读者购书热线：(010)88379203	**封面无防伪标均为盗版**



本书是基于北京斑步至伟科技有限公司 Hadoop 云计算一体机[搭配 PC(个人计算机)]，由北京林业大学工学院与该公司共建的 Bamboo 云计算与物联网实验室共同编写而成，主要介绍如何搭建 Hadoop 平台并在其上面开发并部署云应用。本书主要由李宁、王东亮编著，北京斑步至伟科技有限公司的工程师对本书中的软件实例开发和验证进行了大量、细致的工作，北京林业大学工学院杨尊昊老师、即将留校参加工作的研究生吴健等人参与了一体机的开发与软件测试，实验室全体成员的共同努力确保了本书的质量。

近几年来云计算经过各大厂商的推广，很多读者对于 Hadoop 和云计算已经有了一定的了解，目前有相当一批人在从事这方面的工作，一些高校也开始有这方面的课程和讲座。

本书主要介绍在“裸机”的状态下，如何搭建 Hadoop 平台。通过由浅入深的介绍，引领学生步入 Hadoop 以及云计算的大门。全书从云计算、Hadoop 以及 Linux 的理论知识开始介绍，到集群的搭建、项目的设计和应用，贯穿了 Hadoop 学习的整个过程。通过学习本书，读者基本上可以熟悉市场上各种云平台开发过程和设计思想。

本书各章节内容主要安排如下：

第 1 篇　理论部分

第 1 章：云计算概念、发展现状和实现机制，目前各大厂商的云计算概述；

第 2 章：Hadoop 简介、架构和核心技术——HDFS 以及 MapReduce；

第 3 章：Linux 操作系统的介绍和命令的使用。

第 2 篇　基础实践部分

第 4 章：CentOS 系统的安装和集群的搭建过程；

第 5 章：熟悉 Hadoop 集群的常用命令并学会使用 Web 方式浏览集群；

第 6 章：Hadoop 内部的管理应用、系统体检权限管理；

第 7 章：基础开发实验：包含 HDFS 上传文件、浏览文件和目录、打开、下载和删除文件；

第 8 章：高级开发实验：包含数据去重、排序、平均成绩和单表关联，都是基于 MapReduce 的。

第 3 篇　项目实训部分

第 9 章：个人存储私有云，主要实现云文件的存储、浏览和删除；

第 10 章：气象数据分析云，主要实现对气象数据的分析和计算给出表格；

第 11 章：微信人物关系云，主要根据上传分析条件来对人物关系进行分析；

第 12 章：云图书馆，主要实现电子图书的上传索引，并且提供查询的功能；

第 13 章：物联网与云计算，主要实现智能分配快递的功能。

由于作者水平有限，本书难免存在一些错误和不足，欢迎大家指正。

编著者

2013 年 5 月

目录

前言
第 1 篇　理论部分 …… 1
第 1 章　云计算理论 …… 2
1.1　云计算的概念 …… 2
1.2　云计算发展现状 …… 2
1.3　网格计算与云计算 …… 5
1.4　云计算的发展环境 …… 7
1.4.1　云计算与 3G 移动通信 …… 7
1.4.2　云计算与物联网 …… 7
1.4.3　云计算与移动互联网 …… 8
1.4.4　云计算与三网融合 …… 9
1.5　各大 IT 厂商云计算平台特点概述 …… 10
1.6　开源云计算系统概述 …… 21
第 2 章　Hadoop 理论 …… 29
2.1　Hadoop 简介 …… 29
2.2　Hadoop 架构 …… 30
2.3　HDFS …… 30
2.3.1　设计思想 …… 30
2.3.2　Namenode 和 Datanode 的划分 …… 31
2.3.3　文件系统操作和 namespace 的关系 …… 31
2.3.4　数据复制 …… 32
2.3.5　文件系统元数据的持久化 …… 32
2.3.6　通信协议 …… 33
2.3.7　健壮性 …… 33
2.3.8　数据组织 …… 34
2.3.9　访问接口 …… 34
2.3.10　空间的回收 …… 35
2.4　分布式数据处理 MapReduce …… 35
第 3 章　Linux 命令操作 …… 38
3.1　Linux 操作系统介绍 …… 38
3.1.1　Linux 操作系统的产生 …… 38
3.1.2　Linux 操作系统的开发模式 …… 38
3.1.3　Linux 操作系统的发展 …… 39
3.2　Linux 操作系统常用的 shell 命令 …… 40
3.2.1　基本命令 …… 40
3.2.2　文件目录操作命令 …… 41
3.2.3　vi 编辑器 …… 43
3.2.4　软件包安装命令 …… 44
第 2 篇　基础实践部分 …… 45
第 4 章　集群搭建 …… 46
4.1　CentOS 操作系统的安装 …… 46
4.1.1　实验目的 …… 46
4.1.2　实验设备 …… 46
4.1.3　实验内容 …… 46
4.1.4　实验步骤 …… 46
4.2　集群搭建 …… 63
4.2.1　实验目的 …… 63
4.2.2　实验设备 …… 63
4.2.3　实验内容 …… 63
4.2.4　实验步骤 …… 63
4.3　获取 Hadoop 安装包 …… 84
4.3.1　实验目的 …… 84
4.3.2　实验设备 …… 84
4.3.3　实验内容 …… 84
4.3.4　实验步骤 …… 85
4.4　启动和关闭 Hadoop 集群 …… 89
4.4.1　实验目的 …… 89
4.4.2　实验设备 …… 89
4.4.3　实验内容 …… 89
4.4.4　实验步骤 …… 89
第 5 章　熟悉 Hadoop 本地集群 …… 93
5.1　熟悉 Hadoop 的一些常用命令 …… 93
5.1.1　实验目的 …… 93
5.1.2　实验设备 …… 93
5.1.3　实验内容 …… 93

5.1.4　实验步骤 ………………………… 93
5.2　使用 distcp 进行并行复制 ……… 97
5.3　Web 浏览 Hadoop 集群 …………… 98
5.3.1　实验目的 ………………………… 98
5.3.2　实验设备 ………………………… 98
5.3.3　实验内容 ………………………… 98
5.3.4　实验步骤 ………………………… 98
5.4　使用 Hadoop 命令归档文件 ……… 99
第 6 章　Hadoop 管理应用 ………………… 102
6.1　系统检查和报告 ………………… 102
6.2　了解 HDFS 的平衡命令 ………… 104
6.3　权限设置 ……………………………… 105
6.4　配额管理 ……………………………… 105
6.5　启用回收站 …………………………… 106
第 7 章　HDFS 实践 …………………………… 107
7.1　使用 HDFS 上传文件 ……………… 107
7.1.1　实验目的 ………………………… 107
7.1.2　实验设备 ………………………… 107
7.1.3　实验内容 ………………………… 107
7.1.4　实验原理 ………………………… 107
7.1.5　实验步骤 ………………………… 109
7.2　使用 HDFS 浏览文件和目录 …… 110
7.2.1　实验目的 ………………………… 110
7.2.2　实验设备 ………………………… 110
7.2.3　实验内容 ………………………… 110
7.2.4　实验原理 ………………………… 110
7.2.5　实验步骤 ………………………… 113
7.3　使用 HDFS 打开、下载和删除文件 ………………………………… 114
7.3.1　实验目的 ………………………… 114
7.3.2　实验设备 ………………………… 114
7.3.3　实验内容 ………………………… 114
7.3.4　实验原理 ………………………… 114
7.3.5　实验步骤 ………………………… 117
第 8 章　MapReduce 实践 ………………… 119
8.1　数据去重实验 ………………………… 119
8.1.1　实验目的 ………………………… 119
8.1.2　实验设备 ………………………… 119
8.1.3　实验内容 ………………………… 119
8.1.4　实验原理 ………………………… 120
8.1.5　实验步骤 ………………………… 122
8.2　数据排序实验 ………………………… 125
8.2.1　实验目的 ………………………… 125
8.2.2　实验设备 ………………………… 125
8.2.3　实验内容 ………………………… 125
8.2.4　实验原理 ………………………… 125
8.2.5　实验步骤 ………………………… 128
8.3　平均成绩实验 ………………………… 131
8.3.1　实验目的 ………………………… 131
8.3.2　实验设备 ………………………… 131
8.3.3　实验内容 ………………………… 131
8.3.4　实验原理 ………………………… 132
8.3.5　实验步骤 ………………………… 134
8.4　单表关联实验 ………………………… 136
8.4.1　实验目的 ………………………… 136
8.4.2　实验设备 ………………………… 136
8.4.3　实验内容 ………………………… 136
8.4.4　实验原理 ………………………… 137
8.4.5　实验步骤 ………………………… 142
第 3 篇　项目实训部分 ………………………… 147
第 9 章　个人存储私有云综合实训 …… 148
9.1　实验目的 ……………………………… 148
9.2　实验设备 ……………………………… 148
9.3　实验内容 ……………………………… 148
9.4　实验原理 ……………………………… 148
9.5　实验步骤 ……………………………… 153
第 10 章　气象数据分析云综合实训 …… 161
10.1　实验目的 ……………………………… 161
10.2　实验设备 ……………………………… 161
10.3　实验内容 ……………………………… 161
10.4　实验原理 ……………………………… 161
10.5　实验步骤 ……………………………… 168
第 11 章　微信人物关系云综合实训 …… 172
11.1　实验目的 ……………………………… 172
11.2　实验设备 ……………………………… 172
11.3　实验内容 ……………………………… 172
11.4　实验原理 ……………………………… 172
11.5　实验步骤 ……………………………… 181

第 12 章　云图书馆实例综合实训 ……… 186
12. 1　实验目的 ……………………………… 186
12. 2　实验设备 ……………………………… 186
12. 3　实验内容 ……………………………… 186
12. 4　实验原理 ……………………………… 186
12. 5　实验步骤 ……………………………… 198
第 13 章　物联网与云计算(快递)实例 ……………………………… 201
13. 1　实验目的 ……………………………… 201
13. 2　实验设备 ……………………………… 201
13. 3　实验内容 ……………………………… 201
13. 4　实验原理 ……………………………… 201
13. 5　实验步骤 ……………………………… 207
参考文献 ………………………………………… 212

第1篇

理论部分

第 1 章　云计算理论

1.1　云计算的概念

云计算(Cloud Computing)首先是基于互联网的，并且是对在互联网上的服务进行使用、增加和交付的，这个资源一般是虚拟化并且动态易扩展的。云其实就是对网络以及互联网资源的一种形象说法。在过去，很多人画图会使用云来表示电信网，后来出现了互联网还有对一些底层的基础架构的形象描述。狭义云计算是指一种 IT 基础设施的交付和使用模式，指通过互联网以按需、易扩展的方式获得所需资源；广义云计算指服务的交付和使用模式，指通过互联网以按需、易扩展的方式获得所需服务。这种服务可以是 IT 资源和应用软件、一切和互联网相关资源，也可是其他种类的服务。它意味着计算能力也可作为一种商品通过互联网进行流通。

什么是云计算？云计算是一种基于互联网的超级计算模式，在远程的数据中心，几万台甚至几千万台计算机和服务器连接成一片。因此，云计算甚至可以让人体验每秒超过 10 万亿次的运算能力，如此强大的运算能力几乎无所不能。用户通过计算机、便携式计算机、手机等方式接入数据中心，按各自的需求进行存储和运算。四款比较成熟而实用的云计算产品如下：IBM 公司的蓝云、亚马逊公司的 Amazon EC2、谷歌公司的 Google App Engine、微软公司的 Windows Azure。

1.2　云计算发展现状

1. 云计算发展现状

云计算作为业界热点，近年来世界各国对于它的研究和应用方兴未艾，许多政府部门和著名公司在研发与应用云计算的过程中作出了大量的工作和努力。

(1) 云计算在国外的发展

云计算与网络密不可分。云计算的原始含义是通过互联网提供计算能力。云计算的起源跟亚马逊和谷歌两个公司有十分密切的关系，它们最早使用到了“Cloud Computing”的表述方式。目前美国公开宣布进入或支持云计算技术开发的业界巨头包括微软、谷歌、IBM、亚马逊、Netsuite、NetApp、Adobe 等公司。

谷歌公司是云计算的提出者。2006 年，谷歌公司启动了“Google101”计划，引导大学生们进行“云”系统的编程开发。多年的搜索引擎技术的积累成果使谷歌公司在云计算技术上处于领先的地位，不仅提供在线应用，还希望发挥自身的数据库系统优势，成为在线应用的统一平台。谷歌公司以发表学术论文的形式公开了其云计算三大法宝：GFS、Map/Reduce 和 BigTable，并在美国和我国等国高校开设云计算编程课程。

微软公司于 2008 年 10 月推出了 Windows Azure 操作系统，这个系统作为微软公司云计

算计划的服务器端操作系统(Cloud OS)为广大开发者提供服务。微软公司拥有全世界数以亿计的 Windows 操作系统用户桌面和浏览器，Azure(蓝天)试图通过在互联网架构上打造新的云计算平台，让 Windows 操作系统由 PC(个人计算机)延伸到“蓝天”上。

IBM 公司从企业内部需求的逐渐上升出发，在 2007 年 11 月提出了“蓝云”计划，推出共有云和私有云的概念。IBM 公司提出私有云解决方案是为减少诸如数据、信息安全等共有云现存的问题，从而抢占企业云计算市场。依托 IBM 公司在服务器领域的传统优势，IBM 公司成为目前惟一一个提供从硬件、软件到服务全部自主生产的公司。

2008 年 7 月，雅虎、惠普和英特尔公司联合宣布将建立全球性的开源云计算研究测试床，称为 Open Cirrus，鼓励开展云计算、服务和数据中心管理等领域中各方面的研究。

苹果公司是云计算领域的一位积极参与者。从近年来推出的 iTunes 服务，到 Mobile Me 服务，到收购在线音乐服务商 Lala，再到最近在美国北卡莱罗纳州投资 10 亿美元建立新数据中心的计划，无不显示其进军云计算领域的巨大决心。

这些国际知名大公司在全世界建造了庞大的云计算中心。譬如：谷歌公司的搜索引擎分布于 200 多个站点、超过 100 万台服务器支撑，而且设施数量正在迅猛增长。

（2）云计算在国内的发展

目前我国云计算的讨论多数集中在早期云计算的概念、技术和模式上。早期的云计算是一种动态的、易扩展的、通过互联网提供虚拟化 IT(信息技术)资源和应用的一种计算模式。用户不需要了解云技术内部的细节，也不必具有云内部的专业知识，更不需要直接参与、投入、建设、维护和控制就能直接按需使用并按用量付费。

2008 年，IBM 公司在无锡建立了我国第一个云计算中心，在北京 IBM 中国创新中心建立了第二个云计算中心——IBM 大中华区云计算中心。2009 年初，在南京建立了国内首个“电子商务云计算中心”。世纪互联推出“CloudEx”产品线，包括完整的互联网主机服务“CloudEx Computing Service”、基于在线存储虚拟化的“CloudEx Storage Service”等云计算服务。

随着云计算的升温，国内的电信运营商也都积极投入到云计算的研究中，以期通过云计算技术促进网络结构的优化和整合，寻找到新的赢利机会和利润增长点，以实现向信息服务企业的转型。中国移动公司推出了“大云”(Big Cloud)云计算基础服务平台，中国电信公司推出了“e 云”云计算平台，中国联通公司则推出了“互联云”平台。

我国企业创造了“云安全”概念，通过网状的大量客户端对网络中软件行为的异常监测，获取互联网中木马、恶意程序的最新信息，在服务器端进行自动分析和处理，再把解决方案分发到客户端。瑞星、趋势等企业都推出了云安全解决方案。

随着云计算的发展，互联网的功能越来越强大，用户可以通过云计算在互联网上处理庞大的数据和获取所需的信息。从云计算的发展现状来看，未来云计算的发展会向构建大规模的能够与应用程序密切结合的底层基础设施的方向发展。不断创建新的云计算应用程序，为用户提供更多、更完善的互联网服务也可作为云计算的一个发展方向。

2. 云计算实现机制

由于云计算分为 IaaS(基础设施即服务)、PaaS(平台即服务)和 SaaS(软件即服务)三种类型，不同的厂家又提供了不同的解决方案，目前还没有一个统一的技术体系结构，对读者了解云计算的原理构成了障碍。为此，本书综合不同厂家的方案，构造了一个供商榷的云计

算体系结构。这个体系结构如图 1-1 所示，它概括了不同解决方案的主要特征，每一种方案或许只实现了其中部分功能，或许也还有部分相对次要功能尚未概括进来。

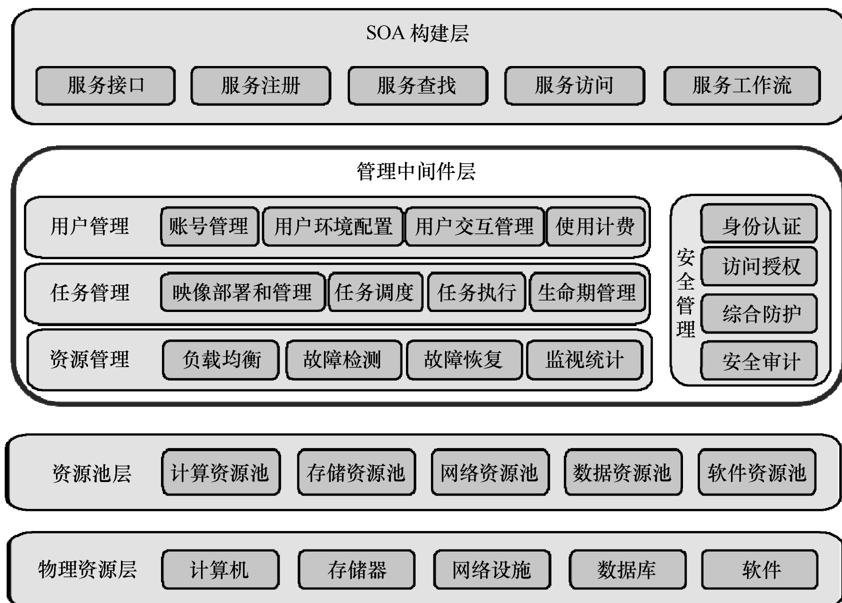


图 1-1　云计算技术体系结构

云计算技术体系结构分为 4 层：物理资源层、资源池层、管理中间件层和 SOA(面向服务的体系结构)构建层，如图 1-1 所示。物理资源层包括计算机、存储器、网络设施、数据库和软件等；资源池层是将大量相同类型的资源构成同构或接近同构的资源池，如计算资源池、数据资源池等。构建资源池层更多是物理资源的集成和管理工作，例如研究在一个标准集装箱的空间如何装下 2000 个服务器、解决散热和故障节点替换的问题并降低能耗；管理中间件层负责对云计算的资源进行管理，并对众多应用任务进行调度，使资源能够高效、安全地为应用提供服务；SOA 构建层将云计算能力封装成标准的 Web Services 服务，并纳入到 SOA 体系进行管理和使用，包括服务注册、查找、访问和构建服务工作流等。管理中间件层和资源池层是云计算技术的最关键部分，SOA 构建层的功能更多地依靠外部设施提供。

云计算的管理中间件层负责资源管理、任务管理、用户管理和安全管理等工作。资源管理负责均衡地使用云资源节点，检测节点的故障并试图恢复或屏蔽之，并对资源的使用情况进行监视统计；任务管理负责执行用户或应用提交的任务，包括完成用户任务映像(Image)的部署和管理、任务调度、任务执行、任务生命期管理等；用户管理是实现云计算商业模式的一个必不可少的环节，包括提供用户交互接口、管理和识别用户身份、创建用户程序的执行环境、对用户的使用进行计费等；安全管理保障云计算设施的整体安全，包括身份认证、访问授权、综合防护和安全审计等。

基于上述体系结构，本书以 IaaS 云计算为例，简述云计算的实现机制，如图 1-2 所示。

用户交互接口向应用以 Web Services 方式提供访问接口，获取用户需求。服务目录是用户可以访问的服务清单。系统管理模块负责管理和分配所有可用的资源，其核心是负载均衡。配置工具负责在分配的节点上准备任务运行环境。监视统计模块负责监视节点的运行状

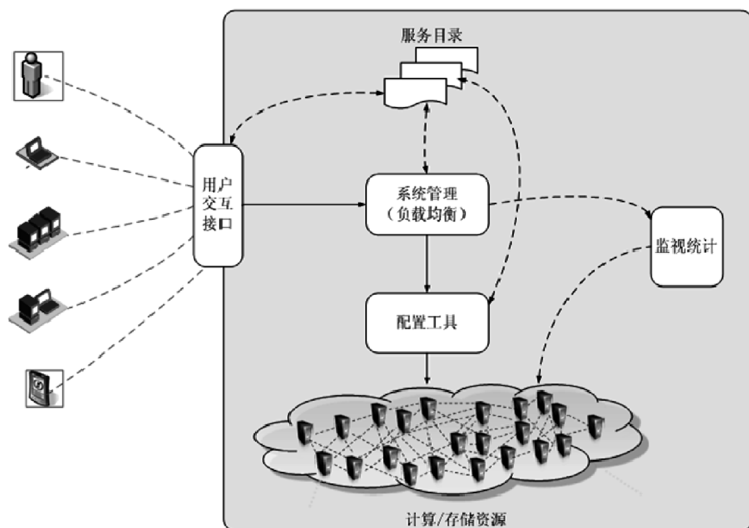


图 1-2　简化的 IaaS 实现机制

态，并完成用户使用节点情况的统计。执行过程并不复杂：用户交互接口允许用户从目录中选取并调用一个服务。该请求传递给系统管理模块后，它将为用户分配恰当的资源，然后调用配置工具来为用户准备运行环境。

1.3　网格计算与云计算

与云计算不同，网格计算已经是一个老词了。当云计算“大红大紫”的时候，人们很少提及网格计算，不过网格计算与云计算有着很深的渊源。

网格计算(Grid Computing)是通过利用大量异构计算机(通常为桌面)的未用资源[CPU(中央处理器)周期和磁盘存储]，将其作为嵌入在分布式电信基础设施中的一个虚拟的计算机集群，为解决大规模的计算问题提供了一个模型。网格计算的焦点放在支持跨管理域计算的能力上，运用平行运算，着重于企业间或跨企业的资源充分运用，共同解决困难的运算任务。这使它与传统的计算机集群或传统的分布式计算相区别。

云计算(Cloud Computing)是一种基于互联网的计算新方式，通过互联网上异构、自治的服务为个人和企业用户提供按需即取的计算。由于资源是在互联网上，而在计算机流程图中，互联网常以一个云状图案来表示，因此可以形象地类比为云，“云”同时也是对底层基础设施的一种抽象概念。

云计算的资源是动态易扩展而且虚拟化的，通过互联网提供。终端用户不需要了解“云”中基础设施的细节，不必具有相应的专业知识，也无需直接进行控制，只关注自己真正需要什么样的资源以及如何通过网络来得到相应的服务。虽然云计算源自平行运算的技术，不脱离网格计算的概念，但是云计算更专注于数据的处理。

云计算其实质还是与以往各类计算机运行的基本过程一样：由输入端输入数据，经数据处理后，再由输出端输出处理后的数据。云计算与网格计算的最大差异在于计算量，云计算大都以单一主机服务用户，主要偏向少量而多次的计算，少次而大量的计算易使资源用尽，

致使其他服务停摆或拒绝服务；网格计算是以多主机来做计算支持，在少次而大量的计算时较为有效率，在此情况下，网格计算域内的计算机资源可互相支持，不会有资源用尽的顾虑。

目前，虽然云计算的概念还没有统一，但云计算已经在人民生活中应用。比如，公交IC(集成电路)卡目前只是使用了预购车票款的加减功能，如果将IC卡上输入更多的持卡人的信息，再将读卡器联系起来，就可以读出某一时段什么年龄的人乘车多、从哪里上车哪里下车、什么线路的车辆拥挤等大量信息。

云计算是网格计算、并行计算和分布式计算的发展，或者说是这些计算机科学概念的商业实现。云计算是虚拟化、效用计算、IaaS、PaaS、SaaS等概念混合演进并跃升的结果。

1. 云计算

使用云计算，企业马上就能大幅提高自己的计算能力，而不需要投资新的基础设施、开展新的培训或者购买新的软件许可证。云计算最适合希望将数据中心基础设施全部外包的中、小型企业，或者希望不用花费高额成本建立更大的数据中心就可获得更高负荷能力的大型企业。不论哪种情况，服务消费者都在互联网上使用所需要的服务并只为所使用的服务付费。

服务消费者不用再守在PC旁边使用PC上的应用程序，或者购买针对特定智能手机、PDA(个人数字助理)及其他设备的版本。消费者不必拥有云中的基础设施、软件或平台，因此降低了前期成本、资本支出和运营成本。消费者也不用关心云中的服务器和网络怎么维护。消费者可以访问任何地方的多台服务器，不需要知道使用的是哪一台服务器以及它们的位置。

2. 网格计算

云计算是从网格计算演化来的，能够随需应变地提供资源。网格计算可以在云中，也可以不在，这取决于什么样的用户在使用它。如果用户是系统管理员或集成商，他们就会关心如何维护云。他们升级、安装和虚拟化服务器与应用程序。如果用户是消费者，就不必关心系统是如何运行的。

网格计算要求软件的使用可以分为多个部分，将程序的片段作为大的系统映像传递给几千个计算机中。网格的一个问题是如果某个节点上的软件片段失效，可能会影响到其他节点上的软件片段。如果这个片段在其他节点上可以使用故障转移组件，那么就可以缓解问题，但是如果软件片段依赖其他软件片段完成一项或多项网格计算任务，那么问题仍然得不到解决。大型系统镜像以及用于操作和维护的相关硬件可能造成很高的资本和运营支出。

3. 异同点

云计算和网格计算都是可伸缩的。可伸缩性是通过独立运行在通过Web服务连接的各种操作系统上的应用程序实例的负载平衡实现的。CPU和网络带宽根据需要分配和回收。系统存储能力根据特定时间的用户数量、实例的数量和传输的数据量进行调整。

两种计算类型都涉及多承租(Multitenancy)和多任务，即很多用户可以执行不同的任务，访问一个或多个应用程序实例。通过大型的用户池共享资源来降低基础设施成本，提高峰值负荷能力。云计算和网格计算都提供了服务水平协议(SLA)以保证可用性，比如99%。如果服务达不到承诺的正常运行时间，消费者将由于数据延迟而得到服务补偿。

亚马逊S3在云中提供了存储和数据检索Web服务。设置在S3中能够存储的对象数量

的最大上限，可以存储只有 1B 的对象，也能存储 5GB 甚至 TB 级的对象。S3 对于对象的每个存储位置使用“桶(Bucket)”作为容器。这些数据采用和亚马逊电子商务网站相同的数据存储基础设施安全地实现存储。

虽然网格中的存储计算非常适合数据密集型存储，但是存储 1B 大小的对象从经济上来说不合适。在数据网格中，分布式数据的数量必须足够大才能发挥最大效益。

计算型网格关注的是计算量非常大的操作。云计算中的亚马逊 Web Services 提供了两种实例：标准和高 CPU。

1.4　云计算的发展环境

1.4.1　云计算与 3G 移动通信

3G 移动通信是第三代移动通信的缩略语。3G 移动通信是指支持高速数据传输的蜂窝移动通信技术，是将无线通信与互联网相结合的新一代通信技术。目前国际电信联盟确定了三个 3G 移动通信标准制式：CDMA2000、WCDMA 和 TD－SCDMA。在我国，中国电信公司、中国联通公司、中国移动公司分别运营这三种不同制式的 3G 移动通信网络。3G 移动通信的代表特征是具有高速数据传输能力，能够提供 2Mbit/s 以上的带宽。因此，3G 移动通信可以支持语音、图像、音乐、视频、网页、电话会议等多种多媒体移动通信业务。

3G 移动通信与云计算是相互依存、相互促进的关系。一方面，3G 移动通信将为云计算带来数以亿计的移动宽带用户。到 2009 年 7 月，全球移动用户已达 44 亿，普及率达 65%。3G 移动通信用户已超过 5 亿，并以惊人的速度增长。2009 年是我国的 3G 元年，当年用户数就超过一千万。这些用户的终端是手机、PDA、便携式计算机等，计算能力与存储空间有限，却有很强的联网能力，对云计算有着天然的需求，将实实在在地支持云计算取得商业成功；另一方面，云计算能给 3G 移动通信用户提供更好的用户体验。云计算有强大的计算能力、接近无限的存储空间，并支撑各种各样的软件和信息服务，能够为 3G 移动通信用户提供前所未有的服务体验。

1.4.2　云计算与物联网

物联网即“物物相连的互联网”。物联网通过大量分散的射频识别、传感器，GPS(全球定位系统)、激光扫描器等小型设备，将感知的信息通过互联网传输到指定的处理设施上进行智能化处理，完成识别、定位、跟踪、监控和管理等工作。笼统地看，物联网属于传感网的范畴。其实，传感器的应用历史悠久而且相当普及。那为什么还提物联网的概念呢？物联网是传感网的一个高级阶段，它通过大量信息感知节点采集信息，通过互联网传输和交换信息，通过强大的计算设施处理信息，然后再对实体世界发出反馈和控制信息。

物联网根据其实质用途可以归结为三种基本应用模式：对象的智能标签、环境监控和对象跟踪与对象的智能控制。物联网基于云计算平台和智能网络，可以依据传感器网络用获取的数据进行决策，改变对象的行为进行控制和反馈。

云计算服务物联网的驱动力有以下三个方面：

1）需求驱动：海量数据的处理在目前技术下有高成本压力。云计算充分利用并合理使

用资源，降低运营成本。

2）技术驱动：IT 与 CT(通信技术)技术融合，推动 IT 架构的升级。云计算的标准逐渐快速发展。

3）政策驱动：政府的低碳经济与节能减排的政策要求。政府高度关注物联网、云计算等基础设施自助发展战略。

物联网具有全面感知、可靠传递和智能处理三个特征，其中智能处理需要对海量的信息进行分析和处理，对物体实施智能化的控制，这就需要信息技术的支持。云计算具有超大规模、虚拟化、多用户、高可靠性、高扩展性等正式物联网规模化、智能化发展所需的技术。

云计算架构在互联网之上，而物联网主要依赖互联网来实现有效延伸，云计算模式可以支撑具有业务一致性的物联网集约运营。因此，很多研究提出了构建基于云计算的物联网运营平台，该平台主要包括云基础设施、云平台、云应用和云管理。依托公众通信网络，以数据中心为核心，通过多介入终端实现泛在接入，面向服务的端到端体系架构。基于云计算模式，实现资源共享和产业协作，提高效率，降低成本，提升服务。

有观点认为云计算是物联网“后端”支撑关键。所谓物联网的“后端”是实现物联网智能化管理目标和价值追求的关键所在。云计算协同信息处理与计算平台对信息处理与决策。实时感应、高度并发、自主协同和涌现效应等特征对物联网“后端”提出了新的挑战，需要有针对性的研究物联网特定的应用集成问题、体系结构以及标准规范，特别是大量高并发时间驱动的应用自动关联和智能协作问题。在互联网计算领域，将软件的实现与运维和用法相关部分(服务)相剥离，并纳入的互联网级基设中，这是大势所趋。而互联网级基设也是云计算、网格计算的本质所在。

物联网与云计算是交互辉映的关系。一方面，物联网的发展也离不开云计算的支撑。从量上看，物联网将使用数量惊人的传感器[如数以亿万计的 RFID(射频识别)、智能尘埃和视频监控等]，采集到的数据量惊人。这些数据需要通过无线传感网、宽带互联网向某些存储和处理设施汇聚，而使用云计算来承载这些任务具有非常显著的性价比优势；从质上看，使用云计算设施对这些数据进行处理、分析、挖掘，可以更加迅速、准确地管理物质世界，从而达到“智慧”的状态，大幅度提高资源利用率和社会生产力水平。可以看出，云计算凭借其强大的处理能力、存储能力和极高的性价比，很自然就会成为物联网的后台支撑平台；另一方面，物联网将成为云计算最大的用户，将为云计算取得更大的商业成功奠定基石。

1.4.3　云计算与移动互联网

互联网与移动通信网是当今最具影响力的两个全球性的网络，移动互联网恰恰融合了两者的发展优势，被称作破坏性创新的云计算，在宽带移动互联网上将成为一种绕不开的趋势。市场调研公司认为，云计算将成为移动世界中的一股爆破理论，最终会成为移动应用的主要运行方式。掌握了云计算核心技术的企业无疑在移动互联网时代可以获得更强的主动性。

移动互联网和云计算是相辅相成的。通过云计算技术，软、硬件获得空前的集约化应用，人们完全可以通过手持一个终端，就能实现传统 PC 能达到的功能。两者在软、硬件设施成本上的极大节约为中、小企业带来了福音，为人们带来了舒适和便捷。

云计算和移动互联网似乎天生就是绝配。手机拥有便捷性和通信能力等众多天然优势，

而计算能力、存储能力弱，虽然各厂商推出的手机正逐渐向智能化演进，但受限于体积和便携性的要求，短时间内手机的处理能力难以和计算机相比。

从这点出发，云计算的特点更能在移动互联网上充分体现，将应用的计算与存储从终端转移到服务器的云端，从而弱化了对移动终端设备的处理需求，成为新业务的发展瓶颈，在云计算下，只要配备功能强大的浏览器就能应用各种新业务，在后台云计算的存储量和计算能力也解决了手机存储量有限和丢失信息等问题。同时，实现了手机移动和固定计算、便携式计算机的协同。对于追求个性化的移动互联网市场来说，云计算的力量十分关键。

移动互联网时代的来临，对用户来讲，最好的体验是淡化有线和无线的概念。在这样的理念下，云计算有望突破各种终端，包括手机、计算机、电视和视听设备等在存储及运算能力上的限制，显示的内容、应用都能保持一致性和同步性。各大 IT 厂商都是利用云计算制定如 IaaS、PaaS 和 SaaS 策略，希望通过互联网的力量，以软件为基准，将无缝的服务提供给移动终端用户。

云计算正从互联网逐渐过渡到移动互联网。目前社交网站越来越火爆，国外的 Facebook 以及国内的人人网、开心网等都是其典型的代表。社交网站运用云计算思维，实现了网站上各种信息的同步更新。沿着这个思路的移动云计算已经出现，如摩托罗拉公司推出的手机解决方案。

如今，随着一些典型的互联网云计算应用，互联网的云与端之间已经形成了平滑对接，而在移动互联网上，云与端之间还需要“管”来沟通它们之间的鸿沟。浏览器或许将成为重要的“管”角色。

云计算对于云与端的两侧都具有传统模式不可比拟的优势。在云一侧，为内部开发者和业务使用者提供更多的服务，提升基础设施的使用效率和资源部署的灵活性；在端一侧，能够迅速部署应用和服务，按需调整业务使用量。从目前云计算的成功案例中可以看出云计算极大地提高了互联网信息技术的性能，具有巨大的计算和成本优势。

1.4.4　云计算与三网融合

所谓的三网融合，是指广播电视网、电信网与互联网的融合，其中互联网是核心。据国务院三网融合领导小组专家组组长、中国工程院副院长邬贺铨估算，三网融合启动的相关产业市场规模达 6880 亿元人民币。其中电信宽带升级、广电双向网络改造、机顶盒产业发展以及基于音频、视频内容的信息服务系统建设的有效投资额为 2490 亿元人民币，可激发和释放的信息服务与终端消费额近 4390 亿元人民币。

三网融合被纳入“十二五”规划，并明确写入《国务院关于加快培育和发展战略性新兴产业的决定》。业内权威专家认为，三网融合的政策持续加码，将推动电信和广电业务相互进入、广电网络整合、网络运营商角色再定位等一系列革命性变化同步加速。仅中国电信一家运营商，其两年内用于宽带升级的投资将达到近 300 亿元人民币。

云计算使计算能力从分散终端向网络综合服务转变，使商业模式从网络设备基础设施向服务转变，从连续计算机资源向连接个人和设备转变。云计算的基础仍然是宽带，其服务手段和服务对象都需要宽带。社会的各种生活、娱乐和就业都对宽带发展提出了高要求，各国也加大了对宽带业务的投入，各厂商也都在加大对宽带业务的研发。

业内专家认为，随着三网融合政策的出台，以及下一代广电网络的出现，云计算不但会

为现有广电和电信产业带来新商机，还会大大拓展相关产业链，使更多企业受益，为云计算提供切实的应用机会。三网融合和下一代广电网项目是要为用户提供多样、便捷的服务。由于云计算能够大大降低数据存储、计算和分发成本，一些以前无法实现的应用，现在都有可能变成现实。云计算完成计算任务，加上物联网等终端应用和3G移动通信的数据信息传输，将三网整合形成一个系统的信息采集、接收和处理的整体。

三网融合和下一代广电网络的最终目标是构建全数据、全融合的国家骨干网络，借助云计算技术，下一代广电网络还会与传统行业相融合，实现诸如远程教育、网络医疗会诊、股票信息、交通查询、精确广告投放等更多应用。有了云计算技术，一些从事传统行业的企业也能搭上三网融合和下一代广电网的快车。例如，传统的GPS厂商只是生产商，而借助于云计算技术，他们可以成为服务性企业，通过增值业务获得更多收入。中国电子协会计算机委员会专家刘鹏认为，云计算在三网融合以及下一代广电网中的应用，涉及数据存储、数据计算、数据再处理、软件开发、数据传输、网络协同等多个方面，因此需要大量不同类型的企业参与其中。

1.5 各大IT厂商云计算平台特点概述

1. 谷歌公司

谷歌公司在互联网搜索方面建立了强大的商业模式，同时也是云计算领域的重要实践者。谷歌公司在其传统搜索引擎、Gmail、Google Web API等产品的基础上针对自己特定的网格应用程序开展起了众多云计算业务，现在不仅提供云服务给众多个人消费者，而且还涉足企业用户，所提供的服务形式包括应用托管服务和企业搜索等。为了支撑其云计算平台，谷歌公司在IT基础架构方面进行了巨大的投入，它在美国的爱荷华州、北卡罗莱纳州和南卡罗莱纳州等州近期已经完成或正在构建全新的数据中心，平均每个造价高达6亿美元。

这里将主要介绍Google App Engine(谷歌应用引擎,GAE)和Google Apps这两个云服务。

(1) GAE

2008年4月，谷歌公司推出了Google App Engine。这是一个可伸缩的Web应用程序云平台，使用户能够在谷歌公司基础设施上构建和托管Web应用程序。GAE提供了一个SDK(软件开发工具包)，使用户可以在本地使用Java或者Python开发和测试Web应用程序，然后部署在远程GAE的生产环境中进行运行、监控和管理。

GAE开始是免费使用，可提供超过500MB的存储空间，以及每月约500万页面浏览量的免费配额。用户可以创建账户，发布应用程序，而无需承担任何费用和风险。当应用程序启用付费后，配额将提高，但用户只需为使用的超过免费水平的资源付费。

GAE基于谷歌公司早就建立起来的底层平台。这个平台包括MapReduce分布式处理技术、GFS(Google File System,谷歌分布式文件系统)和分布式数据库BigTable。其中，MapReduce API提供Map(映射)和Reduce(化简)处理，GFS和BigTable提供数据存取。

(2) MapReduce分布式处理技术

为了简化分布式编程模式，谷歌公司设计了适合于大规模并行数据处理的编程模型——MapReduce，并将其用于自身的搜索引擎系统。该模型用于大规模数据集(大于1TB)的并行运算，使应用程序编写人员只需要将精力放在应用程序本身上，而关于计算机群的可靠性、

可扩展性等问题则交由平台来处理。它可以实现应用程序和底层分布式处理机制的隔离，比如数据分布、调度和容错等，并且尽可能地将计算指令分配在那些保存分布式文件系统的数据节点上，以减少网格的负载。这种模型的核心思想是将需要运算的问题拆解成 Map 和 Reduce 这两个简单的步骤。首先通过 Map 程序将数据切成很多运算块，然后分配给大量不同的计算机处理，最后通过 Reduce 程序将结果合成，输出用户需要的结果。这种编程模型适用于海量数据输入和数据统计等可切分工作。

（3） GFS

GFS 是一个可扩展、结构化的分布式文件系统，它基于 Linux 操作系统普通的文件系统（如 Ext3 等），将多台计算机上的存储空间统一管理起来，可以支持大型、分布式大数据量的读写操作，其容错性较强。在 GFS 中，数据以 64MB 的数据块为存储单位，分布存放到不同的计算机上；每份数据至少在三台计算机上存在副本，并且会保证数据间的同步；为了节约空间，对大量文本型的数据进行压缩保存。

（4） BigTable

基于内存的结构化数据的存储系统。不像关系型数据库（比如 DB2），该系统对事务的支持能力很有限，但其扩展性较好。它支持以表的形式来操作数据，提供了高效查询。还支持查询结果的排序，包括按多属性进行排序。它通过事务分组来控制对写操作的事务。谷歌公司的很多应用，比如 Gmail、Google Maps 都把数据存储在 BigTable 上。

（5） Google Apps

除了提供 GAE 云平台，谷歌公司还提供了很多基于 SaaS 的云应用。Google Apps 提供的东西包括基于 Web 的文档、电子数据表以及其他生产性应用服务。Google Apps 是免费提供给客户使用的，当然谷歌公司也提供收费的高级版本的服务（大约每年 50 美元）。截至目前，已经有 50 多万家组织注册了 Google Apps，整个 Google Apps 的用户数量大约达到了 1000 万。

典型的谷歌公司云计算应用程序就是 Google Docs 服务。由于借鉴了异步网络数据传输的 Web2.0 技术，此类应用程序给予用户全新的界面感受以及更强大的多用户交互能力。Google Docs 是一个给予 Web 的工具，它具有和微软公司 Office 软件相近的编辑界面，由一套简单易用的文档权限管理，而且还可以记录下所有用户对文档所做的修改。Google Docs 的这些功能令它非常适合于网上共享和协作编辑文档。Google Docs 甚至可以用于监控责任清晰、目标明确的项目进度。当前，Google Docs 已经推出了文档编辑、电子表格、幻灯片演示和日程管理等多个功能的编辑模块，能够替代微软公司 Office 软件相应的部分功能。值得注意的是，通过这种云计算方式形成的应用程序非常适合于多个用户进行共享以及协同编辑，为一个小组的人员进行共同创作带来了很大的方便。

2. 亚马逊（Amazon）公司

亚马逊公司是美国一家电子商务网站，也是美国最大的在线零售商，其网络上的销售额大约是美国最大的办公用品零售商 Staples 公司的 3 倍。亚马逊公司在所有布云厂商中是比较特殊的一家，因为它并不是传统意义上的 IT 厂商，但却在 IT 领域的云计算上做得风生水起，成为业界公认的云计算先行者之一。

亚马逊网上书店成立于 1995 年。作为全球电子商务的成功代表，该公司投入了巨大的研发力量和资金来建设自己的数据中心，并且建立了跨越全球的硬件和软件基础设施，来支持自己的互联网业务。他们希望自己的计算中心可以被别人使用，于是将那些基础设施的组

件模块化并且出租。这样亚马逊公司就从一家纯粹的网络书店或者商店变成了云计算服务商、一家具有高科技性质的服务公司，开始为个人以及企业提供云计算服务。

亚马逊公司的云计算服务 AWS（Amazon Web Services）主要包含四个核心服务：Simple storage Services、EC2（Elastic Compute Cloud）、SQS（Simple Queue Service）以及 Simple DB。

（1）Simple storage Services

2006 年 3 月，亚马逊公司首先推出的云计算服务是简单存储服务，它实现了 IaaS 的存储云的功能，并且作为公共存储云服务提供给个人和企业用户使用。

亚马逊公司的简单存储服务 S3 是一种可扩展、高速、低成本的基于 Web 的服务，主要用于文档、图片、影像以及其他应用程序数据的在线备份和存档。S3 允许上传、存储和下载 1B ~ 5GB 大小的文件或对象等非结构化数据。亚马逊公司并没有限制用户可存储的项目的数量。据亚马逊公司网络服务网页报告，S3 是用最低限度的功能设置来设计的，并且开发商通过它可以更容易地使用网络规模计算。

在 S3 中，用户数量存储在多个数据中心的冗余服务器上。它采用一个简单的基于 Web 的界面并且使用密钥来验证用户身份。用户可以选择设置自己的数据位私有数据或者公开数据，并且可以在存储之前对自己的数据进行加密该存储服务按照月租金的形式进行服务付费，同时用户还需要为相应的网络流量进行付费。用户可以在任何地方通过网络从亚马逊网站获取自己的数据。在美国和在欧洲提供服务的费用差异是微小的；在欧洲提供 1000 次的网络存取服务，只比在美国贵大约 2000 美分。

亚马逊公司的 S3 实际上是一个互联网上的大网盘。它没有目录和文件名，只是一个大空间，用户可以在上面存储和提取自己的非结构化数据。可以认为 S3 的结构实际上是一个平面文件系统。然而，在 S3 中除了可以处理数据，也可以处理对象。S3 中的对象可以看作三位一体：关键字、数值和元数据。关键字是该对象的名称，数值是具体的内容，元数据是一组描述对象信息的“键值”对。对象的名称可以是 3 ~ 255 个字符，但是不能采用网址的格式，例如“192. 168. 1. 1”是不符合格式要求的。

（2）EC2

EC2 是一个让用户可以租用云计算机运行所需应用的系统。EC2 借由提供 Web 服务的方式让用户可以弹性地运行自己的亚马逊机器镜像（AMI）文件，用户将可以在这个虚拟机上运行任何自己想要的软件或应用程序。

用户可以随时创建、运行、终止自己的虚拟服务器，使用多少时间算多少钱，也因此这个系统是“弹性”使用的。EC2 让用户可以控制运行虚拟服务器的主机地理位置，这可以让延迟还有备援性最高。例如，为了让系统维护时间最短，用户可以在每个时区都运行自己的虚拟服务器。亚马逊公司以 AWS 的品牌提供 EC2 的服务。

2006 年 8 月，亚马逊公司推出了其最重要的云计算服务，也就是 AWS 现有业务中最大的一片云：EC2。它实现了 IaaS 的计算云的功能，并且作为公共云服务提供给个人和企业用户使用。企业需要计算服务时，不再需要自行购买服务器，而是可以改向亚马逊公司租用，按使用付费。亚马逊公司现有约 4 万台服务器，分布在北美洲和欧洲用以支持 EC2 服务。

亚马逊公司的 EC2 建立在其自己公司内部的大规模集群计算的平台之上，而用户可以通过 EC2 的实例是一些正在运行的虚拟机，每个实例代表一台正在运行中的虚拟机。对于提供给某一个用户的虚拟机，该用户具有完整的访问权限，包括针对该虚拟机的管理员用户

权限。在 EC2 中的每一个计算实例都具有一个内部的 IP 和一个外部的 IP 地址进行数据通信，以获得数据通信的最好性能。用户利用分配给自己的弹性 IP 地址来分配自己的运行实例，使得建立在 EC2 上的服务系统能够为外部提供服务。客户端通过 SOAP（简单对象访问协议）over HTTPS（安全超文本传输协议）来实现与 EC2 内部的实例进行交互，保证远端连接的安全性，以解决用户数据在传输过程中的安全问题。

在用户使用模式上，用户可以首先创建包括操作系统、应用程序和配置设置在内的 AMI 也可使用亚马逊公司预先提供的 AMI 将该机器映像上传至 S3 并注册 EC2，接着创建一个亚马逊机器映像认证符（AMI ID），最后调用亚马逊公司的应用程序编程接口（API），对 AMI 进行使用和管理。AMI 实际上就是虚拟机模板，用户可以使用它来完成任何工作，例如运行数据库服务器、构建网站、提供外部网页服务，甚至可以出租自己具有特色的 AMI 而获得收益。用户所拥有的多个 AMI 可以通过通信而彼此合作，就像当年的集群计算服务平台一样。EC2 的处理能力可以实时增减，至少可以相当于一台虚拟机的处理能力，多至 1000 台以上的处理水平。除了可以提供完整的计算资源外，EC2 还有一些其他的重要管理功能：

1）亚马逊云监测是一项监控 AWS 云资源的网络服务，使亚马逊公司产品的用户可以了解到资源使用、操作性能和总体需求状况，包括 CPU 的使用、磁盘读写和网络流量等指标；

2）自动测量允许 EC2 的容量根据要求增大、减小，保证 EC2 在流量高峰时增容以维持其性能，在容量较低时减容以节约成本，此特性对于使用率波动频繁的程序来说尤其适用；

3）弹性负荷调节，在各 EC2 计算实例之间分配流量，允许程序出错，它能够在资源池中探测出运行不正常的实例，并引导信息流通过正常实例进行，直到不正常实例被修复，使得用户可以使用云资源来进行简单和自动的监控、测量和流量控制，从而帮助用户可以更好地控制他们的 AWS 资源，创造出性能更优、弹性更强、耗费更低的设计。

EC2 的付费方式原则上是按照其计算和所消耗的网络资源收取，但也有针对某些特殊功能的额外服务收费。虚拟机的收费是根据虚拟机的能力进行计算的，因此，实际上用户租用的是虚拟机的计算能力。付费方式则由用户的使用状况决定，即用户仅需要为自己所使用的计算平台实例付费，运行结束后计费也随之结束。在 EC2 中，提供了三种不同能力的虚拟机实例，具有不同的收费价格。例如，据亚马逊公司网络服务网页报告，其中默认的也是最小的运行实例是 1.7GB 内存 1 个 EC2 的计算单元（一个虚拟的计算核对应的计算单元），具有 160GB 的虚拟机内置存储容量，是一个 32 位的计算平台，收费标准为 10 美分/h。在当前的计算平台中，还有两种性能更加优异的虚拟机实例可供使用，相应的价格也更昂贵一些。其他的网络资源（例如 IP 地址和网络服务流量等）消耗的计费也有相应的计费原则。例如内部传输和外部传输的计费原则就不同。额外增值服务收费主要针对某些特殊的功能服务，比如亚马逊云监测和弹性负荷调节。

亚马逊公司通过提供 EC2，把计算、存储和应用作为服务提供。通过物理资源的共享，节约了单一用户的使用成本。通过提供更多的安全机制和可靠性机制，减少了小规模软件开发人员对于集群系统的维护，并且按使用来收费的方式更具成本优势。同时，按用量收费的模式可以帮助扩展软件的销售通路，例如，ORACLE 和 IBM 公司就相继宣布用户可以在 EC2 中运行自己的各项软件产品，按量收费。

（3）SQS

2007 年 7 月，亚马逊公司推出了简单队列服务(SQS)，这项服务使托管主机可以存储计算机之间发送的消息。通过使用 SQS，开发人员可以开发分布式应用程序，并在它们中间用一种安全、灵活和可靠的方式通信，而无需考虑消息丢失的问题。通过这种服务方式解决消息异步传输问题，即使消息的接收方还没有启动也没有关系，服务内部会缓存，而一旦有消息接收，组件被启动运行，队列服务就将消息提交给相应的运行模块进行处理。此外，SQS 服务队列可以被命名并制定访问权限，来决定谁有权读/写队列，并且提供了内置的功能来避免死锁的发生，或用来处理当两个接受者同时访问相同消息的情况。目前，消息只可以是文字，并且长度必须小于 8KB。

任何连入互联网的机器都可以从一个亚马逊队列中读取或者发送内容。队列的接收者可以在不同时间、不同位置接收队列中的数据。队列必须为这种消息传递服务进行付费，通常依据消息的个数以及消息传递的大小进行计算。例如每 10000 个消息收 1 美分。每 GB 的数据传输从 10 美分到 18 美分不等。

(4) SDB

同 S3 专为非结构化的数据块(比如文件)设计不一样，SDB(Simple DB)是一个快速的可伸缩实时数据引擎和查询框架，是为复杂的结构化数据建立的。基于 AWS 的应用程序，可以用它轻松地存储和获取结构化数据。它能够与其他 AWS 很好地协作。跟其他云计算一样，也是只需根据使用量为服务付费，而且还提供一定的免费使用量。

SDB 数据库不是像 DB2 或者 MySQL 那样的关系数据库。按照产品描述，SDB 是“一个对结构化数据实时查询的 Web 服务”。SDB 是使用轻量级并且很容易掌握的查询语言实现的数据库，但它支持大部分可以能会需要的数据库操作(查询、获取、插入和删除等)，并且很容易增长。例如，拥有的域可高达 100 个，每个域中又可以成长到 10GB 和安置多达 2.5 亿个属性。而且不必担心数据会随着数据库的增长而分布到多个磁盘上去。此外，SDB 为支持“实时”(快速周转)查询特性，专门优化了设计。例如，为确保快速查询响应，当数据项被放置在数据库中时，所有属性将自动索引编号。

3. 微软(Microsoft)公司

微软公司是目前全球最大的计算机软件提供商。在云计算已经成为全球 IT 产业共同应和的、主流的声音时，微软公司也走在了时代的前沿。经过多年的积淀和持续探索，微软公司正式发布了一些列称为 S + S(软件 + 服务)的云计算产品和服务。

下面将主要介绍微软公司的公共云服务 Windows Azure 和 Microsoft Live。

(1) Windows Azure

2008 年，在微软公司开发者大会上，微软公司发布了一个全新的云计算平台——Azure Services Platform(www.azure.com)。这是一个基于微软公司数据中心的 PaaS 平台(见图1-3)，提供了一个在线的基于 Windows 操作系统系列产品的开发、存储和服务代管等服务的环境。微软公司的 Azure 平台直接瞄准微软公司产品开发人员，对于每天使用 C#和 SQL Server 的人来说非常亲切。Azure 根据用量定价，使用 Azure 服务越多，价格越高。Azure 可以根据计算时间(CPU 使用)、宽带(包括进和出)和存储计费，也可以根据事务(例如 GET 和 PUT)收费。

Azure 的底层是数据中心中数量庞大的 64 位 Windows 服务器。Windows Azure 通过底层的网状控制器(Fabric Controller，跟 VMware 的 Virtual Center 有着十分相似的功能)将这些服



图 1-3　微软公司 Azure 平台示意

务器有效地组织起来，给前端的应用提供计算和存储能力，并保证其可靠性。它可以看作一个在线的操作系统环境，统治了整个数据中心的运算资源，用户可以很方便地调用这些资源，来执行各种应用程序。此外，操作系统升级、维护等也可以在系统不宕机的情况下自动完成。

同时，微软公司还提供了一套基于 Visual Studio 的 Azure 工具，可供开发者在计算机上开发、模拟和测试 Azure 平台上的应用程序。通过 Azure 工具的发布按钮，开发者能将 ASP. NET 等应用程序直接部署到 Azure 平台上。这样，开发人员就能够缩短开发时间，降低成本，并在熟悉的 Windows 操作系统开发环境及统一的变成模型基础上衍生出新的应用和服务。

Azure 平台目前推出了五项托管服务，包括 . NET 应用服务、SQL 服务、SharePoint 服务、Dynamics CRM 服务以及 Live 服务等，用以帮助客户建立云计算的应用，或将现有的业务拓展到云端。

1）. NET 服务：最初被命名为 BizTalk 服务，它由访问控制、服务总线和工作流三个模块组成。. NET 服务提供了一个基础架构，使用户可以不必一遍一遍开发重复的功能和基础设施来构建基于互联网的分布应用，就可以初步实现互联网服务总线的一些功能。

2）SQL 服务：是一个云计算平台的数据库，构建在企业级的 SQL Server 数据库和 Windows 服务器之上。SQL 服务提供了一系列丰富的集成服务，可以对数据进行查询、搜索、同步、报告和分析之类的操作。数据可以存储在各种设备上，从数据中心最大的服务器一直到桌面计算机和移动设备，用户可以控制数据而不用管数据存储在哪里。另外，SQL 服务可以为应用程序提供较高级别的安全性、可靠性和伸缩性，减少管理和开发应用程序的时间和成本。

3）SharePoint 服务：提供协作服务。通过使用协作特性，企业内的用户可以轻松创建、管理和构建他们的协作 Web 站点，并让这些站点为整个企业所利用，通过这种协作和快速开发的服务建立更强的客户关系。

4）Dynamics CRM 服务：是一个完全集成的客户关系管理系统，提供类似 Salesforce 的应用级服务。通过该服务，用户可以从第一次接触客户开始，在整个购买和售后流程中创建并维护清晰、明了的客户数据；可以强化和改进公司的销售、营销和服务流程，提供快速、灵活且经济、实惠的解决方案；还可以帮助用户在日常业务处理过程中获得持续和显著的改进。

5）Live 服务：以用户为中心，提供诸如联系人信息、博客和图片等服务。微软公司将 Windows Live 的很多功能和资源，通过 Live 服务封装以后提供给软件厂商和开发人员使用。通过 Live 服务，可以存储和管理 Windows Live 用户的信息和联系人，将 Live Mesh 中的文件和应用同步到用户的不同设备上去。

（2）Microsoft Live

Microsoft Live 是微软公司推出的网格托管的云应用服务，主要包括 Office Live 和 Windows Live 两个系列的产品，下面将分别加以介绍。

1）Office Live。Office Live 为小型企业和信息工作者提供了一组新的基于互联网的软件服务，可以帮助他们建立、扩展和管理其在线业务。这些用户可以通过 Web 设计、咨询服务和 Office Live 服务来获得新的收入来源。

Office Live 的服务主要如下：

① Office Web Apps：是一组 Web 版的 Office 应用，它包括 Web 版的 Word、Excel、PowerPoint 和 One Note。任何用户通过浏览器无需安装任何客户端软件即可使用这些 Office 应用，同时还可以共享文件，与其他人协作。Office WebApps 不仅可以通过微软公司的在线服务提供，比如 Hotmail，Docs. com 等，也可以通过 SharePoint 软件运行于私有云上。

② Office 365：它免费提供最基本的服务项目，并同时提供付费的升级服务项目，包括更大的在线空间/存储容量、Store Manager 以及免费的广告管理工具来直接管理通过微软公司 adCenter 注册的广告等。

③ Office Live Add－in：能够将 Office 客户端与 Office Live 整合起来，利用 Microsoft Office 强大的本地处理能力以及 Office Live 提供的在线服务，在本地用 Microsoft Office 打开、编辑和保存 Office Live 上的文档。目前 Office Live Add－in 支持 Word、Excel 和 PowerPoint。

2）Windows Live。Windows Live 是微软公司推出的一项即时沟通工具，也提供照片、空间和邮件等在线服务。著名的 Windows Live Messenger（以前叫 MSN）就是 Windows Live 中的一个产品。Windows Live 可以与包括 Office Live Workspace 和 Office Live Small Business 在内的 Office Live 合并成一套服务，并有一个统一的门户提供给用户。

4. Salesforce 公司

成立于 1999 年的 Salesforce 公司，是全球按需（按需应用、按需付费）客户关系管理（CRM）解决方案的云平台提供商，也是云计算的积极倡导者之一。目前，其在全球的 CRM 付费企业用户数达到 64. 4 万。下面我们将简单介绍 Salesforce 公司的 CRM 云计算服务和用于扩展的 Force. com 平台（见图 1-4）。

（1）Salesforce. com CRM

秉承通过简单网站提供企业应用程序的软件租用的理念，Salesforce 针对中小企业推出了基于互联网的 CRM 服务（www. salesforce. com），开辟了一种新的软件应用模式：通过互联网使用企业应用软件。Salesforce. com CRM 作为基于云计算模式的 CRM，除了提供定制灵活



图 1-4 Force. com 平台

的客户管理和销售管理功能，也充分利用了云计算模式的特性：多租户使用模型、灵活付费模式，以及容量弹性伸缩。用户可以在线开发、配置、运行和监控 CRM 系统；Salesforce. com 的云平台则负责根据负载对服务进行资源调整。

（2）Force. com

Force. com 是 Salesforce 公司基于 PaaS 的公共云计算服务。利用这个平台，可以简单地构建、购买和运行业务应用程序，进一步将计算资源抽象得更加贴近业务。Force. com 可以运行企业资源规划(ERP)、人力资源、供应链、资产跟踪、合同管理以及自定义应用程序。

Force. com 包括一组集成的工具盒应用程序服务，ISV 和公司 IT 部门可以使用它构建业务应用程序，并在提供 Salesforce. com CRM 应用程序的相同基础架构上运行该业务应用程序，包括需要构建企业管理应用的相关内容，而且在单包装中运行应用程序。它包括数据库、集成、业务逻辑、报表、用户界面以及移动服务，这些都通过云服务在 Salesforce 的多租户平台上运行，可避免在复杂的服务器和软件商上浪费宝贵的 IT 资源。图 1-5 显示了构成 Force. com 的主要模块。



图 1-5 构成 Force. com 的主要模块

5. VMware 公司

虚拟化技术是伴随着计算机的出现而产生和发展的，虚拟化意味着对计算机资源的抽象。在云计算概念提出后，虚拟化技术可以用来对数据中心的各种资源进行虚拟化和管理，在物理服务器上虚拟出多个操作系统和应用程序，以便更好地利用计算资源。因此虚拟化技术已经成为构建云计算环境的一项关键技术。作为 X86 体系结构虚拟化技术的代表，VMware 公司基于已有的虚拟化技术优势，面向云计算推出了一系列解决方案和新的技术，尤其是其推出的面向服务器的虚拟机产品 vSphere，号称云计算的首款操作系统。

VMware 公司云计算战略：VMware 公司基于已有的虚拟化技术和优势，提供了云基础架构及管理、云应用平台和终端用户计算等多个层次上的解决方案，主要支持企业级组织机构利用服务器虚拟化技术，实现从目前的数据中心向云计算环境转变。这里根据 VMware 云计算解决方案的三层架构简要介绍 VMware 面向云计算的产品和技术，如图 1-6 所示。

VMware 公司云战略三层架构如下：

1）云基础架构及管理层（IaaS）：云基础架构及管理层由数据中心与云基础架构、安全产品、基础架构和运营管理三大部分组成。数据中心和基础架构是 VMware 公司云计算解决方案的基石。在这一层 VMware 公司的主要思路是通过虚拟化技术将数据中心转变为云计算基础架构，然后通过 VMware 虚拟化提供自助部署和调配的功能，企业可以创建私有云，将 IT 基础架构作为服务来交付使用。面向 IaaS 层的主要产品包括 VMware vSphere 系列和 VMware Server 系列，后者为免费版本，性能不如 vSphere。

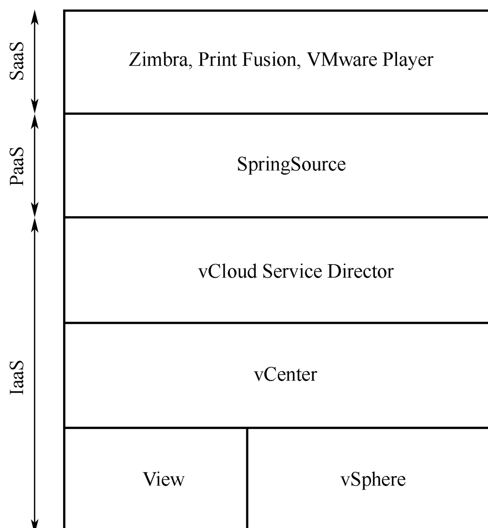


图 1-6　VMware 云计算三层架构

2）云应用平台层（PaaS）：在 PaaS 层，VMware 公司通过收购 SpringSource 来构建基于云的应用开发平台，用于满足用户在云计算模式与环境下开发相应的应用。SpringSource 框架能通过动态、一致的基础架构满足各种企业和 Web 应用的需要，以及简化新应用程序开发的开发者工具和功能。因此，VMware 公司的云应用平台以 SpringSource 应用和 VMware vSphere 为基础，采用高级消息队列协议（AMQP），具有无缝扩展的弹性数据管理技术和跨物理/虚拟环境可见性的性能监控和应用管理机制，并能实现私有云和公有云之间的迁移。

3）终端用户计算解决方案：桌面虚拟化产品。

在 SaaS 层，VMware 公司的定位还不是特别明确，主要是基于桌面和应用程序虚拟化，提供了 VMware ThinApp、VMware Workstation、VMware Fusion、Zimbra、VMware Player、VMware 移动虚拟平台（MVP）及 VMware ACE 等产品。

VMware 的云计算解决方案的重点在于对数据中心等基础架构的虚拟化，因此在 IaaS 层上 VMware 的工作较多，所以这里重点介绍 VMware vSphere 结构、vCloud Service Director 和 VMware View。

1）VMware vSphere 架构。VMware 公司在原来的 VMware Infrastructure 3（以下简称 VI3）基础上推出的 VMware vSphere 被称为业界首款云计算操作系统。VMware vSphere 主要包括两个部分：一是虚拟化管理器 VMM 部分，即 VMware ESX 4；二是用于整合和管理 VMM 的 VMware vCenter。其架构如图 1-7 所示。

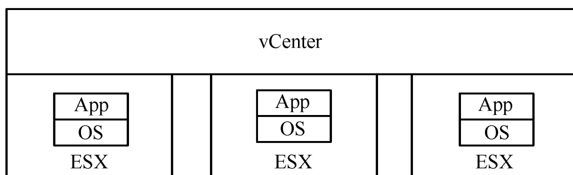


图 1-7　VMware vSphere 架构

虚拟化从结构上可以分为寄居架构和裸金属（Bare Metal）架构：寄居架构指的是在操作系统的层面上进行虚拟机实现，VMware 公司开发的 VMware Workstation 系列就属于寄居架

构；裸金属架构是在计算机硬件上直接进行虚拟化，是架构在计算机硬件和操作系统之间的虚拟化。通过裸金属架构的虚拟化，计算机硬件直接被切割成若干个虚拟机，然后在这些虚拟机上再进行各自的系统和应用程序的安装。这样一来，虚拟机的底层是虚拟出来的 CPU、内存等计算机硬件资源，而不是操作系统，虚拟机之间完全独立。

vSphere 的底层就是 VMware 公司推出的虚拟机 ESX Server。通过 ESX 虚拟化数据中心服务器，将数据中心转换为云计算基础架构，满足 IT 组织利用内部和外部资源、低成本地提供云服务的能力。ESX Serve 属于裸金属架构的虚拟机。ESX Serve 直接安装在服务器硬件上，在硬件和操作系统之间插入了一个稳固的虚拟化层。ESX Serve 讲一个物理服务器划分为多个安全、可移植的虚拟机，这些虚拟机在同一个物理机服务器上运行。每个虚拟机都呈现为一个完成的系统(具有处理器、内存、网络、存储器和 BIOS)，因此 Windows、Linux、Solaris 和 NetWare 操作系统和软件应用程序都可以在虚拟机中运行，无需进行任何修改。

ESX Serve 是向 IT 环境提供虚拟化的分布式服务的基础。ESX Server 最新的版本是 VMware ESX 4，和之前的 VMware ESX 3. 5/3 相比，在功能和特点上有很多更新和扩展，其中最大的区别在于 VMware ESX 4 只支持 64bit 运行模式，只能安装在支持 64 位计算的 X86 物理服务器上。除了 ESX Server，VMware 公司还推出了精简版的 ESXi Server，ESXi Server 与 ESX Server 的最大区别在于 ESXi Server 去除了 Service Console。

VMware ESX 4 主要功能体现在如下三个方面：

① 基础架构服务：即虚拟化管理器(VMM)功能，是整个产品的基础。通过在物理机上的虚拟层可以抽象处理器、内存和 I/O 等资源来运行多个虚拟机。虚拟机能支持高达 8 个虚拟 CPU 和 256GB 内存；还支持热添加功能，可以热添加虚拟 CPU、内存和网络设备，满足应用程序无缝扩展的功能。

② 增强型的基础架构服务：在基础架构服务外，VMware ESX 4 还提供了能增强网络和存储 I/O 性能的 VMDirectPath、能减少存储空间使用的 VSorage Thin Provisioning 和 Linked Clone 等增强的功能。

③ 应用程序服务：VMware ESX 4 提供了 vCenter Agent，用于向 vCenter 上传本机的管理和性能信息，根据 vCenter 的指示协助 vMotion。

云管理平台 vCenter：vCenter 作为管理节点控制和整合属于其域的 vSphere 主机，既可以安装在物理机的操作系统上，也可以安装在虚拟机的操作系统上(官方推荐)。从实现方式上看，它是基于 Java 技术的，后台连接自带的微软公司 SQL Server Express，也可以使用 Oracle 数据库，并可以使用其“链接模式”集成多个 vCenter 支持大量用户的访问。在通信方面，它通过 vSphere 主机内部自带 vCenter Server Agent 与 ESX Server 进行联系，并提供 API 供外部程序和 vCenter 客户端调用。在扩展方面，它支持很多第三方的插件。

vCenter 包括以下六项基本功能：

① 资源和虚拟机的清单管理。该功能可以列出和管理 vCenter 管理域内所有的资源(如存储、网络、CPU 和内存等)和虚拟机。

② 任务调度。支持定时任务或者及时任务(如 vMotion)，满足各个任务之间不出现抢占资源或者冲突的要求。

③ 日志管理。用于记录任务和事件的日志。

④ 警告和事件管理。使用户可以及时获知系统出现的新情况。

⑤ 虚拟机部署。通过部署向导，上传 vApp 和虚拟磁盘等，部署虚拟机。

⑥ 主机和虚拟机的设置。用户可以修改一些主机和虚拟机的主要配置，而且还能对那些非常底层的特性进行设置，比如是否开启硬件辅助虚拟化。

vCenter 还有以下七个方面的高级功能：

① 动态迁移 vSphere 提供了 vMotion 和 Storage vMotion 技术，分别满足虚拟机和虚拟磁盘的热迁移。

② 资源优化。VMware 公司的分布式资源调度(Distributed Resource Scheduler,DRS)技术，通过将虚拟机从资源紧张的主机迁移到资源剩余的主机等方式来实现资源优化，使得每个虚拟机都能找到合适的位置。

③ 安全方面。VMware 公司推出两大虚拟机安全技术：一是推出 VMware API，对虚拟机进行安全扫描检测病毒和恶意软件；二是推出 VMware Shield Zones，主要起到防火墙的作用，可监视、记录和组织 vSphere 主机内部或集群中主机之间和虚拟机之前的流量。

④ 容错。VMware Fault Tolerance 是 VMware 提供的虚拟机容灾技术。

⑤ 高可用性。VMware High Availability 技术通过心跳机制来检测虚拟机的运行状态，并通过在其他主机上重启无响应的虚拟机的方式来保障系统的可用性。

⑥ 备份。VMware 采用了加固备份技术(VMware Consolidated Backup,VMCB)，在没有安装 Agent 时对多个虚拟机进行集中备份。

⑦ 应用部署。VMware vApp 基于开放式虚拟化格式(Open Virtualization Format,OVF)协议，将应用程序转化为自描述和自管理型实体，以方便部署和降低管理开支。

2）底层架构服务 vCloud Service Director。vSphere 的主要目的是将底层物理资源进行虚拟和管理，但仅安装了 vSphere 的数据中心并不能称之为云平台。VMware 公司通过 vCloud Service Director，在 vSphere 架构上利用一系列虚拟技术，提供连接企业虚拟环境与私有云的接口和自动化管理工具，通过运行 vCloud Express 与外部服务商无缝地连接，向外提供云 IaaS 服务。vCloud Service Director 使 IT 部门能够通过基于 Web 的门户向用户开放虚拟数据中心，并定义和开放能部署在虚拟数据中心的 IT 服务目录。

vCloud Service Director 早期被称为 Redwood 项目，目前该产品的资料 VMware 公司向外公布得不多。vCloud Service Director 的架构如图 1-8 所示，它具有数据库与管理资源池的服务总线通信的功能。另外，利用基于 VMware vCloud Service Director 提供云服务的 VMware 公司服务提供商体系，可以将数据中心容量扩展到安全、兼容的公共云中，并像管理企业的私有云一样方便地管理。利用基于策略的用户控制技术和 VMware vShield 安全技术，保持多租户环境的安全性和可控性。以虚拟数据中心的形式向内部组织高效地提供资源，提高整合率并简化管理，降低成本。以渐进方式实现云计算，利用现有投资和开放标准，以保证云之间的互操作性和应用程序的可移植性。

3）虚拟桌面产品 VMware View。VMware View 是 VMware 公司的桌面虚拟化产品，通过 VMware View 能够在一台普通的物理服务器上虚拟出很多台虚拟桌面(Virtual Desktop)供远端用户使用。

VMware View 的主要部件包括：

① View Connection Server：View 连接服务器，View 客户端通过它连接 View 代理，将接收的远程桌面用户请求重定向到相应的虚拟桌面、物理桌面或终端服务器。

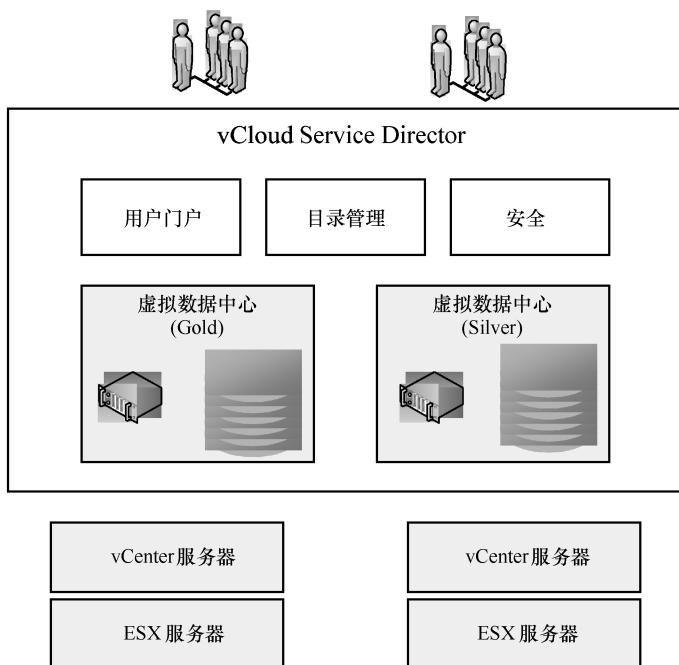


图 1-8　vCloud Service Director 的架构

② View Manager Security Server：View 安全连接服务器，是可选组件。

③ View Administrator Interface：View 管理接口程序，用于配置 View Connection Server、部署和管理虚拟桌面、控制用户身份验证。

④ View 代理：View 代理程序，安装在虚拟桌面依托的虚拟机、物理机或终端服务器上，安装后提供服务，可由 View Manager Server 管理。该代理具备多种功能，如打印、远程 USB 运行和单点登录。因为在 VMware vSphere Server 提供的虚拟机不包括声卡、USB 接口支持等，必须安装该软件，才可以将 VMware vSphere Server 提供的虚拟机连接到 View Client 计算机的相应设备上并显示、应用在客户端。

⑤ View Client：View 客户端程序，安装在需要使用“虚拟桌面”的计算机上通过它可以与 View Connection Server 通信，从而允许用户连接到虚拟桌面。

⑥ View Client with Offline Desktop：也是 View 客户端程序，但该软件支持 View 脱机桌面，可以让用户“下载”vSphere Server 中的虚拟机到“本地”运行。

⑦ View Composer：安装在 vCenter Server 上的软件服务，可以通过 View Manager 使用“克隆链接”的虚拟机，这是 View 4 提供的新功能，在以前的 View 3 版本中，每个虚拟机桌面智能使用一个独立的虚拟机，而添加该组件后，可以让虚拟桌面使用“克隆链接”的虚拟机，这不仅提高了部署虚拟桌面的速度，也减少了 vSphere 的空间占用。

1.6　开源云计算系统概述

自 Hadoop、Eucalyptus 这两种典型的开源云计算系统问世后，利用这些开源项目提供的各种工具，研究者可以在实验室用很低的成本，在由普通机器构成的集群系统中模拟出近似

商业云的环境，这也掀起了开源云计算系统研究的热潮，本节将介绍一些其他新兴的开源云计算系统，带领读者进入多姿多彩的开源云计算世界。

本节对一些开源云计算系统作概念性介绍，重点介绍 Cassandra、Hive 和 VoltDB 等开源云计算系统。

1. Cassandra

Cassandra 是一套高度可扩展、最终一致、分布式的结构化键值存储系统。它结合了 Dynamo 的分布技术和谷歌公司的 BigTable 数据模型，更好地满足了海量数据存储的需求，解决了应用与关系数据库之间存在的非依赖关系。同时，Cassandra 变更垂直扩展为水平扩展，相比其他典型的键值数据存储模型，Cassandra 提供了更为丰富的功能。

Cassandra 最初由 Avinash Lakshman（亚马逊公司 Dynamo 的作者之一）和 Prashant Malik（Facebook 公司工程师）在 Facebook 公司设计开发，2008 年 Facebook 把它贡献给了开源社区，从某种程度上来说，可以把 Cassandra 看成是 Dynamo 的升级版本 2.0，或者是 Dynamo 与 BigTable 的集合。Cassandra 的设计目标如下：

1）高可用性；

2）最终一致性；

3）动态可伸缩；

4）可以动态调整一致性/持久性与延时；

5）节点管理要保持低开销；

6）最小化管理开销。

考虑到 Cassandra 的设计目标，在一致性、可用性和分区容忍度（CAP 理论）的折中问题上，Cassandra 选择了 AP（即可用性和分区容忍度）。针对基本的一致性哈希分布不均匀且不能根据节点能力强弱分配的缺点，Dynamo 让每个点管理环中的多个位置，而 Cassandra 让负载轻的节点在环上移动来实现负载均衡。在应用中，Cassandra 表现出了以下六个突出的特点：

1）模式灵活。使用 Cassandra 就像文档存储，可以在系统运行时随意地添加或者移除字段。而不必提前去解决记录中的字段。特别是在大型部署上将极大地提升效率。

2）真正的可扩展性。Cassandra 是纯粹意义上的水平扩展。为给集群添加更多容量，可以动态添加节点而不必重启任何进程、改变应用查询或手动迁移任何数据。

3）多数据中心识别。通过调整节点的布局避免某一个数据中心出现故障，一个备用的数据中心将至少有每条记录的完全复制。

4）范围查询。可以设置键的范围来进行键值查询。

5）列表数据结构。在混合模式可以将超级列添加到五维，这使得每个用户索引将变得非常方便。

6）分布式写操作。可以在任何地方、任何时间集中读或者写任何数据，并且不会有任何单点失败。

当前，Cassandra 系统正在得到迅猛发展，社交网站巨头 Facebook 公司采用 Cassandra 存储 Inbox，Twitter、WebEx 和 Digg 公司也做了大量向 Cassandra 的迁移工作。

2. Hive

Hive 起源于 Facebook 公司，是一个基于 Hadoop 的数据仓库工具，同时也是 Hadoop 的

一个主要子项目。Hive 提供了一系列的工具，可以用来进行数据的提取、转换盒加载（ETL），同时可以实现对 Hadoop 中大规模数据的存储、查询和分析。Hive 定义了一种简单的类似 SQL——HiveQL。HiveQL 使熟悉 SQL 的用户可以方便地在 Hadoop 中查询数据。同时 Hive 还有很强的灵活性，没有将用户限制在一个框架中，主要表现在当 Hive 内建的 Mapper 和 Reducer 不能满足用户的需求时，用户可以通过 Map/Reduce 将自己开发的 Mapper 和 Reducer 加入到 Hive，以满足用户特殊的需求。

Hive 没有定义所谓的 Hive 格式的数据，可以在 Thift 上很好地工作，控制分隔符，设置可以自己定义数据格式。

作为 Hadoop 的主要子项目，Hive 秉承开源的精神，在不断地发展中，不断有新的特性加入其中。现在已经增加和将要增加的一些新特性如下：

1）增加了用于收集分区和列的水平系统计数值的命令；

2）支持在 Partition 级别去更改 Bucket 的数量；

3）在 Hive 中实现检索；

4）为 Hive 增加并发模型；

5）支持在两个或者两个以上列中的差别选择；

6）利用 bloom 过滤器提高连接的效果；

7）建立 Hive 的授权结构和认证结构；

8）在 Hive 中使用位图检索。

3. VoltDB

VoltDB 是 Mike Stonebreaker（Postgres 和 Ingres 的联合创始人）领导团队开发的下一代开源数据库管理系统。在 VoltDB 中，所有事务被实现为 Java 存储过程。VoltDB 大幅降低了服务器资源开销，单节点每秒数据处理远远高于其他数据库管理系统。它可以在现有的廉价服务器集群上实现每秒数百万次数据处理。不同于 NoSQL 的 key - value 存储，VoltDB 能使用 SQL 存取，并支持传统数据库的 ACID（原子性、一致性、隔离性、持久性）模型。

在 VoltDB 内部，采用并行单线程从而保证了事务的一致性和高效率，由于减少了锁的管理、资源管理等开销，VoltDB 具有极高的处理效率和速度。VoltDB 开发人员的测试表明，与一个优化过的传统数据库管理系统相比，VoltDB 可以达到后者 45 倍的事务处理速度。VoltDB 还具有以下一些传统数据库不能同时具有的优点：

1）可以达到几乎线性的扩展；

2）满足 ACID 特性；

3）提供相比传统数据库好很多的性能；

4）使用 SQL 作为数据库接口。

VoltDB 支持多节点并行事务处理，理论上不存在节点上限，目前 VoltDB 开发人员大的测试集群可以达到 20 个节点。同时，VoltDB 还存在不少限制，主要包括以下四种：

1）不支持动态修改 Schema；

2）增加节点需要停止服务；

3）不支持 xDBC；

4）AdHoc 查询性能不优化。

4. Enomaly ECP

Enomaly ECP 的全称是 Enomaly 弹性计算平台（Enomaly's Elastic Compiting Platform），之前的名称是 Enomalism。它是一个可编程的虚拟云架构，企业可以利用 ECP 将自己的数据中心和云计算服务商的设备连接起来，简化不同数据中心间虚拟机的转移及各种云应用发布的过程。

Enomaly ECP 是一个开放源代码项目，提供了一个功能类似于 EC2 的云计算框架。Enomaly ECP 基于 Linux 操作系统，同时支持 Xen 和 KVM（Kernel Virtual Machine，kernel 虚拟机）。与其他解决方案不同的是，Enomaly ECP 提供了一个基于 TurboGears Web 应用程序框架和 Python 的软件栈。Enomaly ECP 架构如图 1-9 所示。

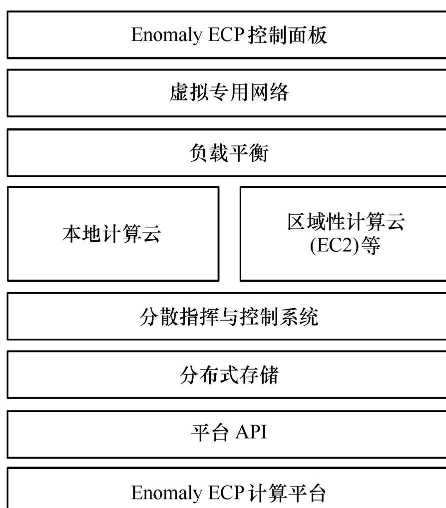


图 1-9　Enomaly ECP 架构

Enomaly ECP 具有以下四个特性：

1）自动供应：ECP 提供自动化的云供应引擎，用于配置、管理和部署成组的虚拟机。

2）灵活性：ECP 能够直接将资源提供给用户的应用程序，并且可以对获得授权使用应用程序者进行限制。

3）可扩展性：ECP 的混合云模型无缝连接使用 ECP 的内部云和外部云提供商。

4）整合现有基础设施：通过 ECP 提供的丰富的 API，用户可以直接管理和配置当前系统。

目前 Enomaly ECP 有两个版本：一个是仍旧免费的社区版（Community Edition）；另一个是提供全方位技术支持的服务提供商版（Service Provider Edition）。

5. Nimbus

Nimbus 是基于网格中间件 Globus 的作品，从最早的 Virtual Workspace 演化而来，提供与 EC2 类似的功能和接口。Nimbus 是一个开源的工具集，它可以把集群部署到 IaaS 云中。Nimbus 通过一整套工具来提供 IaaS 形式的云计算解决方案。Nimbus 属于科学云（Science Clouds）的一部分，该项目创建的最初目的是为了搭建一个科学试验用的云计算平台，但现在其应用已经超出了这个范围，开始涉及其他领域。Nimbus 具有如下四个特点：

1）具有两套 Web 服务接口——Amazon EC2 WSDL 和符合网格社区 WSRF 规范的接口；

2）可以执行基于 Xen 管理程序；

3）可以使用如 PBS 或 SGE 调度器去调度虚拟机；

4）定义了一个可扩展架构，用户可以根据项目的需求进行定制。

Nimbus 的架构如图 1-10 所示。Nimbus 的架构比较复杂，涉及的新概念较多，这里只对其中四个重要的概念进行简单解释。

1）标准客户端（Reference Client）：以命令行的方式访问服务，全面支持 WSRF 前台的各种特性；

2）WSRF（Web Services Resource Framework，Web 服务资源框架）；

3）RM API：RM 是 Resource Management 的缩略语，也就是资源管理，RMAPI 即资源管

理 API；

4）工作区（Workspace）：实际上就是一个计算节点。

不过并不是所有的平台都必须使用图 1-10 中的全部组件，图 1-10 中的各个组件之间可以采取不同的组合方式选择使用。

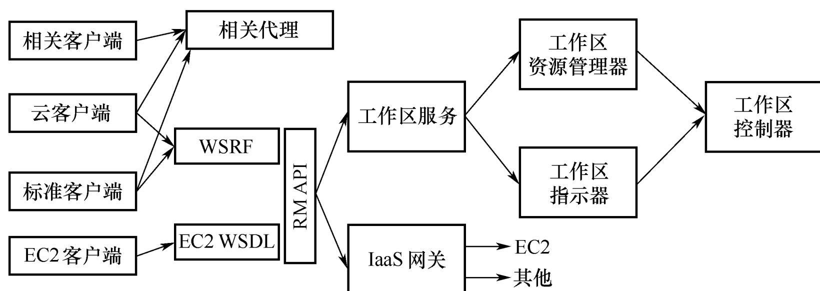


图 1-10　Nimbus 的构架

6. Sector and Sphere

Yunhong Gu 等人设计实现了 Sector and Sphere 云计算平台。Sector and Sphere 是用 C ++ 语言编写的，包括 Sector 和 Sphere 两个部分。Sector 是部署在广域网上的分布式存储系统，它为了使系统有高可靠性和可用性而采用了自动的文件副本冗余方式，已经用于 Sloan 数字巡天系统。Sphere 是建立在 Sector 之上的计算服务，它为用户编写分布式密集型数据应用提供了简单的编程接口。

Sector 采用主/从服务器模式，其架构如图 1-11 所示。安全服务器维护用户的账户、密码、文件访问信息和授权的从节点的 IP 地址；主服务器维护存储在系统中的文件的元数据，控制所有的从节点的运行，同时和安全服务器进行通信来验证从节点、客户服务器和用户；从节点用来存储数据，并对 Sector 客户端的请求进行处理。

Sphere 是以 Sector 为基础构建的计算云，提供大规模数据的分布式处理。Sphere 的基本数据处理模型如图 1-12 所示。

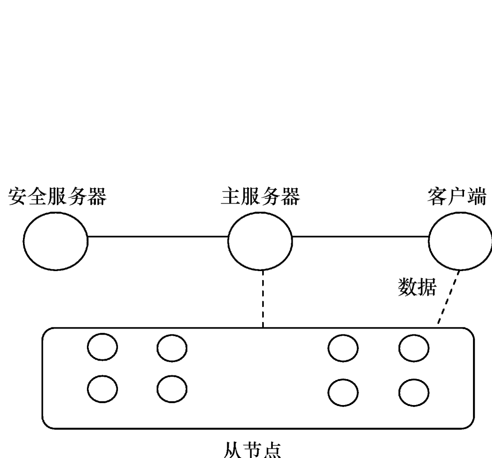


图 1-11　Sector 架构

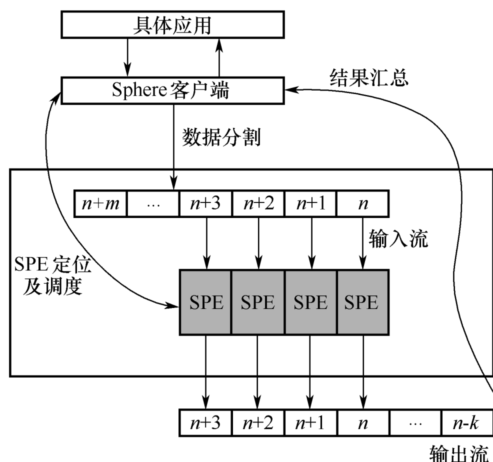


图 1-12　Sphere 的基本数据处理模型

针对不同的应用会有不同的数据，Sphere 统一将它们以数据流的形式输入。为了便于大规模地并行计算，首先要对数据进行分割，分割后的数据交给 SPE 执行。SPE(Sphere Processing Engine,Sphere 处理引擎)是 Sphere 的基本运算单元。除了进行数据处理外，SPE 还能起到负载平衡的作用，因为一般情况下数据量远大于 SPE 数量，当前负载较重的 SPE 能继续处理的数据就较少，反之则较多，如此就实现了系统的负载平衡。SPE 处理后的结果可以作为最终结果以输出流形式输出，也可以作为下一个处理过程的输入。Sphere 客户端为编写分布式应用程序的开发者提供了一系列的 API 包，开发者可以使用这些 API 包来初始化输入流、加载处理函数库、启动 Sphere 进程和读取处理结果。具体流程如下：

1）主服务器接收到 Sphere 数据处理的客户端请求后，主服务器向客户端发送一个可用的从节点列表；

2）客户端选择一些或者所有从节点，让 SPE 在其上运行；

3）客户端与 SPE 建立 UDT 连接；

4）流处理函数被发送给每个 SPE，并存储在从节点上；

5）SPE 打开动态库并获得各种处理函数。

图 1-13 展示了使用两个 Sphere 进程执行分布式排序的过程。第一阶段采用哈希函数扫描全部的数据流，把每个元素放置到相应的桶中。第二阶段使用 SPE 对每个桶排序。在图 1-13 中，首先将需要排序的数据进行散列(Hash)处理，将它们比较均匀地分布到有序的位置上，最好能使每个位置所包含的数据量大致相同。比如要排序的是数字，并且这些数字都在 500 之内，那么我们就可以将大于 300 的全部散列到位置 A，150～300 的散列到位置 B，剩下的散列到位置 C。这一步完成后再次利用 SPE 对各个位置进行排序，具体的排序方法可以自行选择。经过这两个过程后即可完成排序，因为此时将 A、B、C 中的数据一次输出就是一个有序数列。

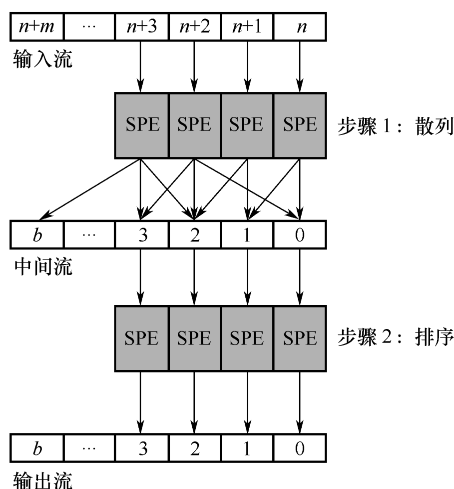


图 1-13　使用两个 Sphere 进程执行分布式排序的过程

这里用一个实例说明 Sphere 的作用。假设有十亿张天文图像存储在 SDSS 中，要从中找出褐矮星的图片，用户需要写一个 findBrownDwarf 的函数从大量图片中找出需要的图片，在这个函数中，把所有图片作为输入，褐矮星图片作为输出，findBrownDwarf(input,output)标准的内部函数如下：

```
for each file F in(SDSS slices)
   for each image I in F
      findBrownDwarf(I,…);
```

使用 Sphere 客户端的 API，伪代码如下：

```
SphereStream sdss;
Sdss. init(/*list of SDSS slices*/);
SphereProcess myproc;
```

Myproc run(sdss," findBrownDwarf");

Myproc. read(result);

其中，sdss 是存储 Sector 文件元数据的流数据结构。

通过使用 Sector 和 Sphere 能够将数据传输速度提升至 Hadoop 的两倍，速度提升的原因之一就是采用 UDT(基于 UDP 的数据传输协议)，这一协议主要是针对极高速网络和大型数据集设计的。

7. abiquo

abiquo 公司推出了一套比较完整的云计算解决方案，它可以帮助用户在各种复杂环境下高效地构建公有、私有或混合云。这套方案主要包括三个部分：abiCloud、abiNtense 和 abiData。三个部分可以单独使用，也可搭配起来使用。

abiCloud 是 abiquo 公司的最重要产品，是一款开源云管理软件，可以创建管理资源并且可以按需扩展。该工具能够以快速、简单和可扩展的方式创建和管理大型、复杂的 IT 基础设施(包括虚拟服务器、网络、应用和存储设备等)。abiCloud 较之同类其他产品的一个主要区别在于其强大的 Web 界面管理，用户可以通过拖曳一个虚拟机来部署一个新的服务。它允许通过 VirtualBox 部署实例，还支持 VMware、KVM 和 Xen 等不同的虚拟机。

abiCloud 目前主要有三个版本：社区版(Community Version)、企业版(Enterprise Version)和 ISP 版(ISP Version,ISP 即互联网服务提供商)。社区版向公众免费提供，企业版则在社区版基础上添加了一些高级特性，而 ISP 版通过扩展企业版的内容来允许 ISP 出售其云计算服务。

abiCloud 的基础构架如图 1-14 所示。

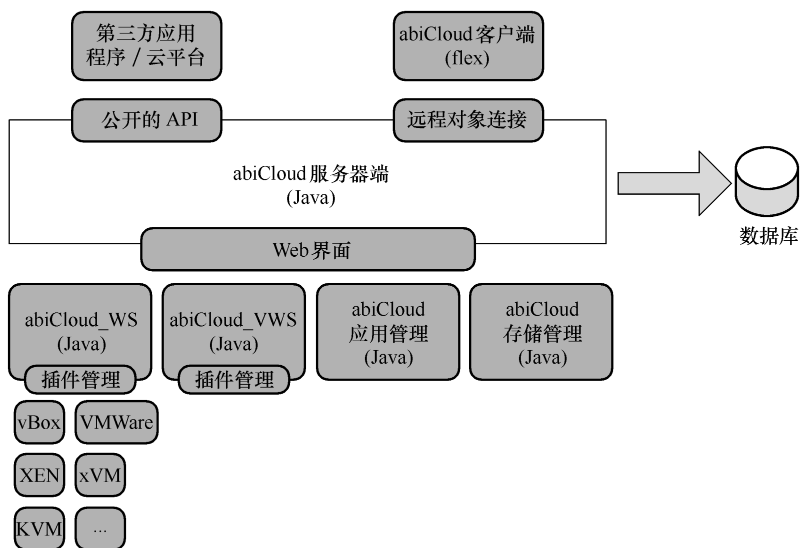


图 1-14　abiCloud 的基本构架

abiCloud 的基本构架清晰明了，其中 abiCloud_WS 是平台的虚拟工厂，主要负责管理各种虚拟化技术。abiCloud_VMS(abiCloud Virtul Monitor System)用来监控虚拟化设备的运行状态。从图 1-14 中可以看出，abiCloud 除了客户端采用了 flex 技术外，其他部分几乎都是由 Java 语言来实现的。和 abiCloud 相比，其他两个产品使用的并不是很多。AbiNtense 通过使

用基于网络的架构，有效地减少了大规模高性能计算的执行时间。abiData 由 Hadoop Common、Hbase 和 Pig 开发而来，它是一个信息管理系统，可以用来搭建分析大量数据的应用，是一种低成本的云存储解决方案。

8. MongoDB

MongoDB 是由 10gen 公司支持的一项开源计划。10gen 公司云平台可用于创建私有云，是类似于 Google App Engine 的一个软件栈，提供与 App Engine 类似的功能，但有一些不同之处，通过 10gen 公司可以使用 Python、JavaScript 和 Ruby 编程语言开发应用程序。该平台还使用沙盒概念隔离应用程序，并且使用自己的应用服务器在 Linux 上构建可靠的环境。

MongoDB 的目标是构建一个基于分布树文件存储系统的数据库，由 C ++ 语言编写。

MongoDB 易于部署、管理和使用，主要设计目标是高性能、可扩展和适当的功能。

MongoDB 主要有以下 6 个特性：

1）易存储对象类型的数据；

2）高性能，特别适合“高容量、值较低”的数据类型；

3）支持动态查询；

4）支持复制和故障恢复；

5）自动处理碎片以支持云计算层次上的扩展性；

6）使用高效的二进制数据存储方式，可以存储包括视频在内的大型数据。

MongoDB 的架构如图 1-15 所示。

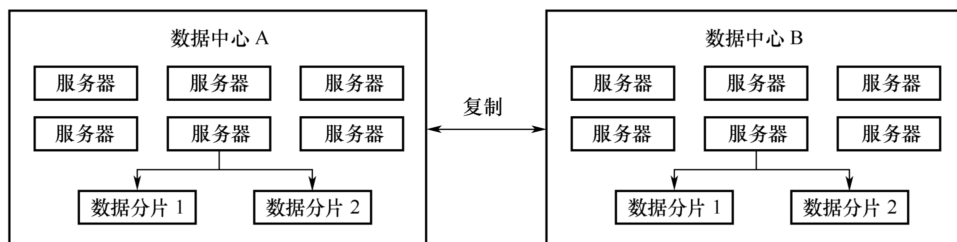


图 1-15　MongoDB 的架构

考虑到系统存在失效的情况，MongoDB 对所存储的数据都进行了备份，将不同的数据副本存储在不同的数据中心。对于存储的数据 MongoDB 又进行了分割，将不同的分片存放在数据中心的不同服务器上，这就解决了大规模数据的存储和查询问题。

图 1-16 所示是 MongoDB 与目前主流的存储方案进行的简单比较。

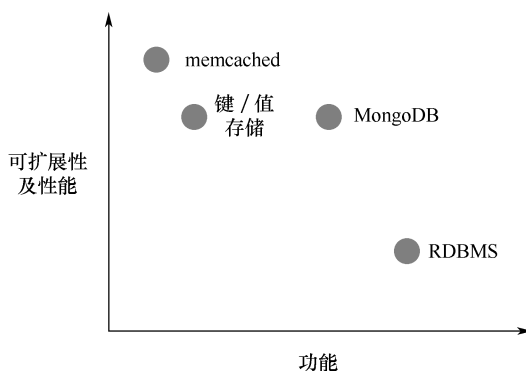


图 1-16　主流存储方案对比

从图 1-16 所示中可以看出，MongoDB 的最大优势在于它的均衡性，MongoDB 可以在功能以及扩展性方面找到一个绝佳的平衡点。其他存储方式要么功能强大但扩展性和性能较差（RDBMS），要么可扩展很好但功能有限（memcached、键/值对方式存储）。

第 2 章　Hadoop 理论

2.1　Hadoop 简介

Hadoop 是一个分布式系统基础架构，由 Apache 基金会开发。用户可以在不了解分布式底层细节的情况下开发分布式程序，充分利用集群的威力进行高速运算和存储。Hadoop 实现了一个分布式文件系统(Hadoop Distributed File System，HDFS)。HDFS 有着高容错性的特点，并且设计用来部署在低廉的(Low-cost)硬件上。而且它提供高传输率(High Throughput)来访问应用程序的数据，适合那些有着超大数据集(Large Data Set)的应用程序。HDFS 放宽了(Relax)POSIX 的要求(Requirements)，这样可以流的形式访问(Streaming Access)文件系统中的数据。

1. Hadoop 名字的起源

Hadoop 这个名字不是一个缩写，它是一个虚构的名字。该项目的创建者，Doug Cutting 如此解释 Hadoop 的得名："这个名字是我孩子给一个棕黄色的大象样子的填充玩具命名的。我的命名标准就是简短，容易发音和拼写，没有太多的意义，并且不会被用于别处。小孩子是这方面的高手。"

2. Hadoop 的起源

Hadoop 由 Apache Software Foundation 公司于 2005 年秋天作为 Lucene 的子项目 Nutch 的一部分正式引入。它最先受到由 Google Lab 开发的 Map/Reduce 和 Google File System(谷歌文件系统，GFS)的启发。2006 年 3 月，Map/Reduce 和 Nutch Distributed File System (Nutch 分布式文件系统，NDFS)分别被纳入称为 Hadoop 的项目中。

Hadoop 是最受欢迎的在互联网上对搜索关键字进行内容分类的工具，但它也可以解决许多要求极大伸缩性的问题。例如，如果要 grep 一个 10TB 的巨型文件，会出现什么情况？在传统的系统上，这将需要很长的时间。但是 Hadoop 在设计时就考虑到这些问题，采用并行执行机制，因此能大大提高效率。

3. 诸多优点

Hadoop 是一个能够对大量数据进行分布式处理的软件框架。但是 Hadoop 是以一种可靠、高效、可伸缩的方式进行处理的。Hadoop 是可靠的，因为它假设计算元素和存储会失败，因此它维护多个工作数据副本，确保能够针对失败的节点重新分布处理。Hadoop 是高效的，因为它以并行的方式工作，通过并行处理加快处理速度。Hadoop 还是可伸缩的，能够处理拍字节(PB)级数据。此外，Hadoop 依赖于社区服务器，因此它的成本比较低，任何人都可以使用。

Hadoop 带有用 Java 语言编写的框架，因此运行在 Linux 操作平台上是非常理想的。Hadoop 上的应用程序也可以使用其他语言编写，比如 C ++ 语言。

2.2　Hadoop 架构

Hadoop 有许多元素构成，如图 2-1 所示。其最底部是 Hadoop Distributed File System(Hadoop 分布式文件系统,HDFS)，它存储 Hadoop 集群中所有存储节点上的文件。HDFS(对于本书)的上一层是 MapReduce 引擎，该引擎由 JobTrackers 和 TaskTrackers 组成。

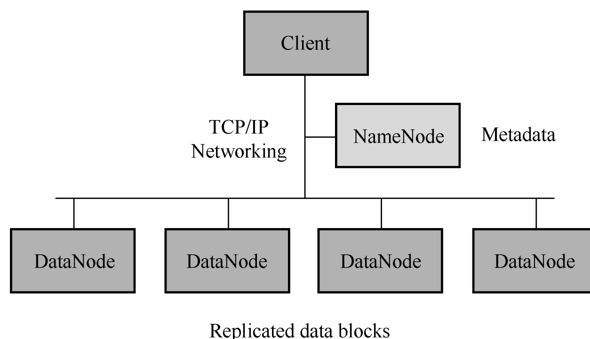


图 2-1　Hadoop 架构

2.3　HDFS

2.3.1　设计思想

1. 硬件失效是“常态事件”，而非“偶然事件”

HDFS 可能是由上千台机器组成(如 Yahoo 的 Hadoop 集群有 4096 个节点)，任何一个组件都有可能一直失效，因此数据的健壮性错误检测和快速、自动恢复是 HDFS 的核心架构目标。

2. 流式数据访问

运行在 HDFS 上的应用和普通的应用不同，需要流式访问它们的数据集。HDFS 的设计中更多地考虑到了数据批处理，而不是用户交互处理。比之数据访问的低延迟问题，更关键的在于数据并发访问的高吞吐量。POSIX 标准设置的很多硬性约束对 HDFS 应用系统不是必需的。为了提高数据的吞吐量，在一些关键方面对 POSIX 的语义做了一些修改。

3. HDFS 应用对文件要求的是 write-one-read-many 访问模型

一个文件经过创建、写、关闭之后就不需要改变。这一假设简化了数据一致性问题，使高吞吐量的数据访问成为可能。典型的如 MapReduce 框架，或者一个 web crawler 应用都很适合这个模型。

4. 移动计算的代价比移动数据的代价低

一个应用请求的计算，离它操作的数据越近就越高效，这在数据达到海量级别的时候更是如此。将计算移动到数据附近，比将数据移动到应用所在显然更好，HDFS 提供给应用这样的接口。

5. 在异构的软硬件平台间的可移植性

2.3.2 Namenode 和 Datanode 的划分

一个 HDFS 集群由一个 Namenode 和一定数目的 Datanode 组成，如图 2-2 所示。

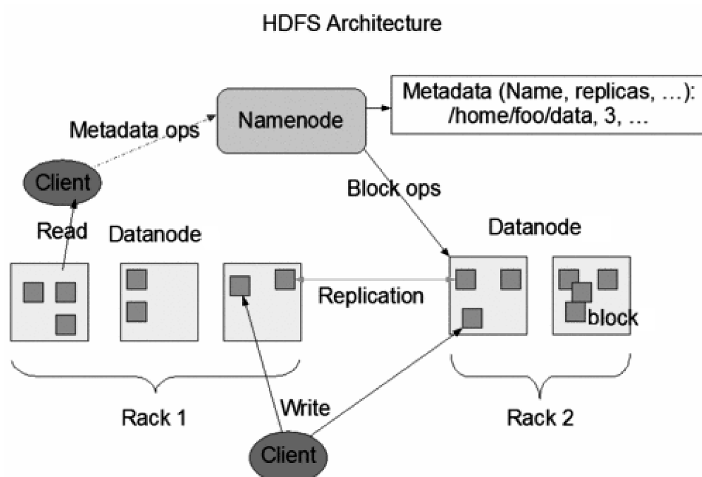


图 2-2　HDFS 集群

Namenode 是一个中心服务器，负责管理文件系统的 namespace 和客户端对文件的访问。

Datanode 在集群中会有多个，一般是一个节点存在一个，负责管理其自身节点上附带的存储。在内部，一个大文件分成一个或多个 block，这些 block 存储在 Datanode 集合里。Namenode 执行文件系统的 namespace 相关操作，例如打开、关闭、重命名文件和目录，同时决定了 block 到具体 Datanode 节点的映射。Datanode 在 Namenode 的指挥下进行 block 的创建、删除和复制。

单一节点的 Namenode 大大简化了系统的架构。Namenode 负责保管和管理所有的 HDFS 元数据，因而在请求 Namenode 得到文件的位置后就不需要通过 Namenode 参与而直接从 Datanode 进行。

为了提高 Namenode 的性能，所有文件的 namespace 数据都在内存中维护，所以就天生存在了由于内存大小的限制导致一个 HDFS 集群提供服务的文件数量的上限。

根据目前的文档，一个元数据(一个 HDFS 文件块)占用 200B，如果是页面抓取的小文件，那么 32GB 内存能承载 1.5 亿个左右的文件存储(有待精确详细测试)。

2.3.3 文件系统操作和 namespace 的关系

HDFS 支持传统的层次型文件组织，与大多数其他文件系统类似，用户可以创建目录，并在其间创建、删除、移动和重命名文件。HDFS 不支持 user quotas 和访问权限，也不支持链接(Link)，不过当前的架构并不排除实现这些特性。Namenode 维护文件系统的 namespace，任何对文件系统 namespace 和文件属性的修改都将被 Namenode 记录下来。应用可以设置 HDFS 保存的文件的副本数目，文件副本的数目称为文件的 replication 因子，这个信息也是由 Namenode 保存。

2.3.4　数据复制

HDFS 被设计成在一个大集群中可以跨机器可靠地存储海量文件。它将每个文件存储成 block 序列，除了最后一个 block，所有的 block 都是同样大小。文件的所有 block 为了容错都会被复制。每个文件的 block 大小和 replication 因子都是可配置的。replication 因子可以在文件创建时配置，以后也可以改变。HDFS 中的文件是 write-one，并且严格要求在任何时候只有一个 writer。Namenode 全权管理 block 的复制，它周期性地从集群中的每个 Datanode 接收心跳包和一个 Blockreport。心跳包的接收表示该 Datanode 节点正常工作，而 Blockreport 包括了该 Datanode 上所有的 block 组成的列表。

1. 副本的存放

副本的存放是 HDFS 可靠性和性能的关键。庞大的 HDFS 实例一般运行在多个机架的计算机形成的集群上，不同机架间的两台机器的通信需要通过交换机，显然通常情况下，同一个机架内的两个节点间的带宽会比不同机架间的两台机器的带宽大。在大多数情况下，replication 因子是 3，HDFS 的存放策略是将一个副本存放在本地机架上的节点上，一个副本放在同一机架上的另一个节点上，最后一个副本放在不同机架上的一个节点上。机架的错误远远比节点的错误少，这个策略不会影响到数据的可靠性和有效性。1/3 的副本在一个节点上，另外 2/3 在一个机架上，其他保存在剩下的机架中，这一策略改进了写的性能。

2. 副本的选择

为了降低整体的带宽消耗和读延时，HDFS 会尽量让 reader 读最近的副本。如果在 reader 的同一个机架上有一个副本，那么就读该副本。如果一个 HDFS 集群跨越多个数据中心，那么 reader 也将首先尝试读本地数据中心的副本。

3. SafeMode

Namenode 启动后会进入一个称为 SafeMode 的特殊状态，处在这个状态的 Namenode 是不会进行数据块的复制的。Namenode 从所有的 Datanode 接收心跳包和 Blockreport。Blockreport 包括了某个 Datanode 所有的数据块列表。每个 block 都有指定的最小数目的副本。当 Namenode 检测确认某个 Datanode 的数据块副本的最小数目，那么该 Datanode 就会被认为是安全的；如果一定百分比(这个参数可配置)的数据块检测确认是安全的，那么 Namenode 将退出 SafeMode 状态，接下来它会确定还有哪些数据块的副本没有达到指定数目，并将这些 block 复制到其他 Datanode。

2.3.5　文件系统元数据的持久化

Namenode 存储 HDFS 的元数据。对于任何对文件元数据产生修改的操作，Namenode 都使用一个称为 Editlog 的事务日志记录下来。例如，在 HDFS 中创建一个文件，Namenode 就会在 Editlog 中插入一条记录来表示；同样，修改文件的 replication 因子也将往 Editlog 插入一条记录。Namenode 在本地 OS 的文件系统中存储这个 Editlog。整个文件系统的 namespace，包括 block 到文件的映射、文件的属性，都存储在称为 FsImage 的文件中，这个文件也是放在 Namenode 所在系统的文件系统上。Namenode 在内存中保存着整个文件系统 namespace 和文件 Blockmap 的映像。这个关键的元数据设计得很紧凑，一般占用 200B 的内存，因而一个

带有 4GB 内存的 Namenode 足够支撑海量的文件和目录。当 Namenode 启动时，它从硬盘中读取 Editlog 和 FsImage，将所有 Editlog 中的事务作用(Apply)在内存中的 FsImage 上，并将这个新版本的 FsImage 从内存中 flush 到硬盘上，然后再 truncate 这个旧的 Editlog，因为这个旧的 Editlog 的事务都已经作用在 FsImage 上了。这个过程称为 checkpoint。在当前实现中，checkpoint 只发生在 Namenode 启动时，在不久的将来我们将实现支持周期性的 checkpoint。Datanode 并不知道关于文件的任何东西，除了将文件中的数据保存在本地的文件系统上。它把每个 HDFS 数据块存储在本地文件系统上隔离的文件中。Datanode 并不在同一个目录创建所有的文件，相反，它用启发式方法来确定每个目录的最佳文件数目，并且在适当的时候创建子目录。在同一个目录创建所有的文件不是最优的选择，因为本地文件系统可能无法高效地在单一目录中支持大量的文件。当一个 Datanode 启动时，它扫描本地文件系统，对这些本地文件产生相应的一个所有 HDFS 数据块的列表，然后发送报告到 Namenode，这个报告就是 Blockre port。

2.3.6 通信协议

所有的 HDFS 通信协议都构建在 TCP/IP 上。客户端通过一个可配置的端口连接到 Namenode，通过 ClientProtocol 与 Namenode 交互。而 Datanode 使用 DatanodeProtocol 与 Namenode 交互。从 ClientProtocol 和 Datanodeprotocol 抽象出一个远程调用(RPC)，在设计上，Namenode 不会主动发起 RPC，而是响应来自客户端和 Datanode 的 RPC 请求。

2.3.7 健壮性

HDFS 的主要目标就是实现在失败情况下的数据存储可靠性。常见的三种失败为 Namenode failures、Datanode failures 和网络分割(Network partitions)。

1. 硬盘数据错误、心跳检测和重新复制

每个 Datanode 节点都向 Namenode 周期性地发送心跳包。网络切割可能导致一部分 Datanode 跟 Namenode 失去联系。Namenode 通过心跳包的缺失检测到这一情况，并将这些 Datanode 标记为 dead，不会将新的 IO 请求发给它们。寄存在 dead Datanode 上的任何数据将不再有效。Datanode 的死亡可能引起一些 block 的副本数目低于指定值，Namenode 不断地跟踪需要复制的 block，在任何需要的情况下启动复制。在下列情况可能需要重新复制：某个 Datanode 节点失效、某个副本遭到损坏、Datanode 上的硬盘错误或者文件的 replication 因子增大。

2. 集群均衡

HDFS 支持数据的均衡计划，如果某个 Datanode 节点上的空闲空间低于特定的临界点，那么就会启动一个计划自动地将数据从一个 Datanode 搬移到空闲的 Datanode。当对某个文件的请求突然增加时，也可能启动一个计划创建该文件新的副本，并分布到集群中以满足应用的要求。这些均衡计划目前还没有实现。

3. 数据完整性

从某个 Datanode 获取的数据块有可能是损坏的，这个损坏可能是由于 Datanode 的存储设备错误、网络错误或者软件 bug 造成的。HDFS 客户端软件实现了 HDFS 文件内容的校验和。某个客户端创建一个新的 HDFS 文件，会计算这个文件每个 block 的校验和，并作为一

个单独的隐藏文件保存这些校验和在同一个 HDFS namespace 下。客户端检索文件内容，它会确认从 Datanode 获取的数据跟相应的校验和文件中的校验和是否匹配，如果不匹配，客户端可以选择从其他 Datanode 获取该 block 的副本。

4. 元数据磁盘错误

FsImage 和 Editlog 是 HDFS 的核心数据结构。如果这些文件损坏了，整个 HDFS 实例都将失效。因而，Namenode 可以配置成支持维护多个 FsImage 和 Editlog 的拷贝。任何对 FsImage 或者 Editlog 的修改，都将同步到它们的副本上。这个同步操作可能会降低 Namenode 每秒能支持处理的 namespace 事务。这个代价是可以接受的，因为 HDFS 是数据密集的，而非元数据密集。当 Namenode 重启时，它总是选取最近的一致的 FsImage 和 Editlog 使用。Namenode 在 HDFS 是单点存在，如果 Namenode 所在的机器错误，手工的干预是必需的。目前，在另一台机器上重启因故障而停止服务的 Namenode 这个功能还没实现。

2.3.8 数据组织

1. 数据块

兼容 HDFS 的应用都是处理大数据集合的。这些应用都是写数据一次，读可以是多次，并且读的速度要满足流式读。HDFS 支持文件的 write-once-read-many。一个典型的 block 大小是 64MB，因而文件总是按照 64MB 切分成 chunk，每个 chunk 存储于不同的 Datanode 上。

2. 数据产生步骤

某个客户端创建文件的请求其实并没有立即发给 Namenode，事实上，HDFS 客户端会将文件数据缓存到本地的一个临时文件中。应用的写被透明地重定向到这个临时文件。当这个临时文件累积的数据超过一个 block 的大小(默认 64MB)，客户端才会联系 Namenode。Namenode 将文件名插入文件系统的层次结构中，并且分配一个数据块给它，然后返回 Datanode 的标识符和目标数据块给客户端。客户端将本地临时文件 flush 到指定的 Datanode 上。当文件关闭时，在临时文件中剩余的没有 flush 的数据也会传输到指定的 Datanode，然后客户端告诉 Namenode 文件已经关闭。此时 Namenode 才将文件创建操作提交到持久存储。如果 Namenode 在文件关闭前关闭了，该文件将丢失。上述方法是通过对 HDFS 上运行的目标应用认真考虑的结果。如果不采用客户端缓存，由于网络速度和网络堵塞会对吞估量造成比较大的影响。

3. 数据块复制

当某个客户端向 HDFS 文件写数据时，一开始是写入本地临时文件，假设该文件的 replication 因子设置为 3，那么客户端会从 Namenode 获取一张 Datanode 列表来存放副本。然后客户端开始向第一个 Datanode 传输数据，第一个 Datanode 一小部分(4kbit)一小部分地接收数据，将每个部分写入本地仓库，并且同时传输该部分到第二个 Datanode 节点。第二个 Datanode 也是这样，边收边传，一小部分一小部分地收，存储在本地仓库，同时传给第三个 Datanode。第三个 Datanode 就仅仅是接收并存储了。这就是流水线式的复制。

2.3.9 访问接口

HDFS 给应用提供了多种访问方式，可以通过 DFSShell 通过命令行与 HDFS 数据进行交互，可以通过 java API 调用，也可以通过 C 语言的封装 API 访问，并且提供了浏览器访问的

方式。正在开发通过 WebDav 协议访问的方式。

2.3.10 空间的回收

1. 文件的删除和恢复

用户或者应用删除某个文件，这个文件并没有立刻从 HDFS 中删除。相反，HDFS 将这个文件 mv 到/trash 目录。当文件还在/trash 目录时，该文件可以被迅速地恢复。文件在/trash 目录中保存的时间是可配置的，超过这个时间，Namenode 就会将/trash 目录中的文件批量从 namespace 中删除。文件的删除，也将释放关联该文件的数据块。并且需要注意的是，在文件被用户删除和 HDFS 空闲空间的增加之间会有一个等待时间延迟。当被删除的文件还保留在/trash 目录中时，如果用户想恢复这个文件，可以检索浏览/trash 目录并检索该文件。/trash 目录仅仅保存被删除文件的最近一次拷贝。/trash 目录与其他文件目录没有什么不同，除了一点：HDFS 在该目录上应用了一个特殊的策略来自动删除文件，目前的默认策略是删除保留超过 6h 的文件，这个策略以后会定义成可配置的接口。

2. replication 因子的减小

当某个文件的 replication 因子减小，Namenode 会选择要删除的过剩的副本。下次心跳检测就将该信息传递给 Datanode，Datanode 就会移除相应的 block 并释放空间，同样，在调用 setReplication 方法和集群中的空闲空间增加之间会有一个时间延迟。

2.4 分布式数据处理 MapReduce

最简单的 MapReduce 应用程序至少包含三个部分：一个 map 函数、一个 reduce 函数和一个 main 函数。main 函数将作业控制和文件输入/输出结合起来。在这点上，Hadoop 提供了大量的接口和抽象类，从而为 Hadoop 应用程序开发人员提供许多工具，可用于调试和性能度量等。

MapReduce 本身就是用于并行处理大数据集的软件框架。MapReduce 的根源是函数性编程中的 map 和 reduce 函数。它由两个可能包含有许多实例(许多 map 和 reduce)的操作组成。map 函数接收一组数据并将其转换为一个键/值对列表，输入域中的每个元素对应一个键/值对。reduce 函数接收 map 函数生成的列表，然后根据它们的键(为每个键生成一个键/值对)缩小键/值对列表。

MapReduce 流程的概念流如图 2-3 所示。

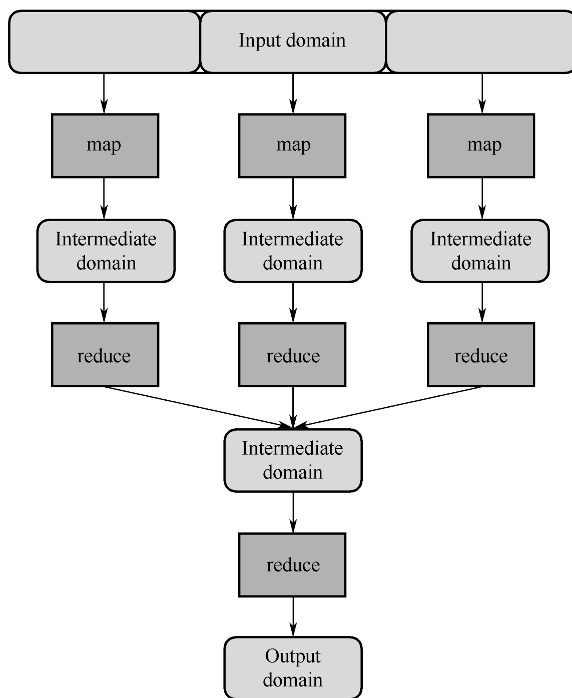


图 2-3 MapReduce 流程的概念流

这里提供一个示例，帮助理解。假设

输入域是 one small step for man，one giant leap for mankind。在这个域上运行 map 函数将得出以下键/值对列表：

(one,1)(small,1)(step,1)(for,1)(man,1)

(one,1)(giant,1)(leap,1)(for,1)(mankind,1)

如果对这个键/值对列表应用 reduce 函数，将得到以下一组键/值对：

(one,2)(small,1)(step,1)(for,2)(man,1)(giant,1)(leap,1)(mankind,1)

结果是对输入域中的单词进行计数，这无疑对处理索引十分有用。但是，现在假设有两个输入域：第一个是 one small step for man；第二个是 one giant leap for mankind。可以在每个域上执行 map 函数和 reduce 函数，然后将这两个键/值对列表应用到另一个 reduce 函数，这时得到与前面一样的结果。换句话说，可以在输入域并行使用相同的操作，得到的结果是一样的，但速度更快。这便是 MapReduce 的“威力”；它的并行功能可在任意数量的系统上使用。图 2-4 以区段和迭代的形式演示了这种思想。

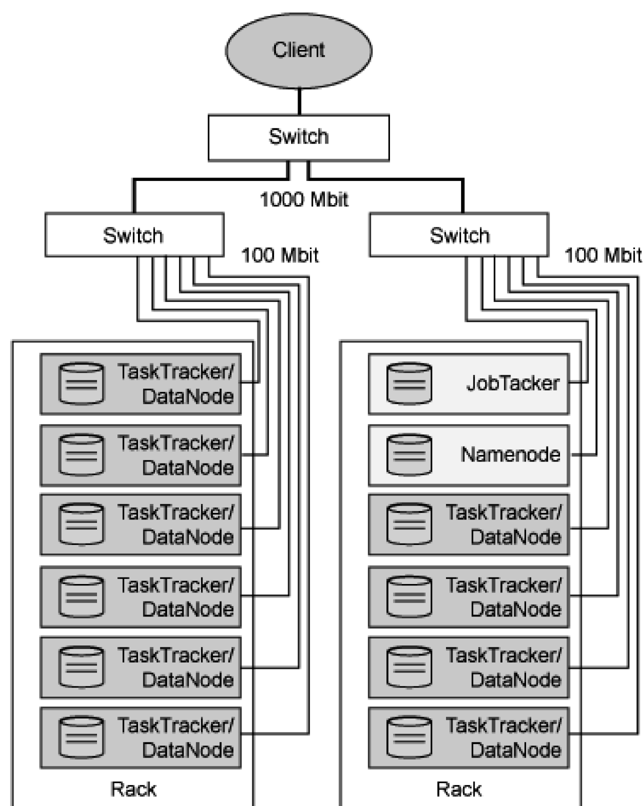


图 2-4 显示处理和存储的物理分布的 Hadoop 集群

现在回到 Hadoop 上，它是如何实现这个功能的？一个代表客户机在单个主系统上启动的 MapReduce 应用程序称为 JobTracker。类似于 NameNode，它是 Hadoop 集群中惟一负责控制 MapReduce 应用程序的系统。在应用程序提交之后，将提供包含在 HDFS 中的输入和输出目录。JobTracker 使用文件块信息(物理量和位置)确定如何创建其他 TaskTracker 从属任务。MapReduce 应用程序被复制到每个出现输入文件块的节点，将为特定节点上的每个文件块创

建一个惟一的从属任务。每个 TaskTracker 将状态和完成信息报告给 JobTracker。图 2-4 显示了一个示例集群中的工作分布。

Hadoop 的这个特点非常重要，因为它并没有将存储移动到某个位置以供处理，而是将处理移动到存储位置。这通过根据集群中的节点数调节处理，因此支持高效的数据处理。

第 3 章　Linux 命令操作

3.1　Linux 操作系统介绍

3.1.1　Linux 操作系统的产生

Linux 操作系统是一种计算机操作系统，是 UNIX 操作系统的一种克隆系统，这是因为 Linux 操作系统和 UNIX 操作系统有着很深的渊源。

在计算机非常昂贵的年代，只有在大学或大型企业中才能够接触到计算机，人们非常希望多个用户能同时连接到一台计算机并同时使用它。于是，计算机科学家开始研究分时系统。分时系统是将 CPU 的运行时间分为很小的时间片，多个用户任务可以通过交替占有时间片的方式实现快速交互使用 CPU。由于时间片是很短的一段时间，以至于每个用户任务、每个用户好像在独占 CPU，独占整个计算机系统。在研究人员的不懈努力下，1969 年，AT&T 公司贝尔实验室开发出了 UNIX 操作系统。

1986 年，芬兰赫尔辛基大学的 Andrew Tanenbaum 教授为了给学生讲授《计算机操作系统》课程，开发出了 Minix 操作系统，这是 UNIX 操作系统的一个变体。1991 年，Andrew Tanenbaum 教授的学生 Linus Torvalds，由于对课堂上使用的 Minix 操作系统不太满意，于是开始在 80386 PC 上试着改进 Minix 操作系统。

1991 年 8 月，Linus Torvalds 在 comp. os. minix 新闻组贴上了以下这段话：“你好，所有使用 Minix 操作系统的人，我正在为 80386(486)AT 做一个免费的操作系统，只是为了爱好，…”。

Linus 最初为自己的这套系统取名为 freax，他将源代码放在了芬兰的一个 FTP 站点上供大家下载。该站点的管理员认为这个系统是 Linus 的 Minix 系统，因此建立了一个名为 Linux 的文件夹来存放它。于是，Linus 的“爱好”就成了今天微软公司的头号对手，功能强大且价格低廉的 Linux 操作系统。

1993 年年底至 1994 年年初，Linux 1. 0 操作系统终于诞生了！

Linux1. 0 操作系统已经是一个功能完备的操作系统，而且内核写得紧凑、高效，可以充分发挥硬件的性能，在 4MB 内存的 80386 机器上也表现得非常好，至今人们还在津津乐道。

3.1.2　Linux 操作系统的开发模式

Linus 于 1991 年 10 月 5 日发布了 Linux 操作系统的第一个版本 Linux 0. 0. 2，并在网络上公布了 Linux 操作系统核心程序的源代码，同时决定以 GPL(大众所有版权，又称 GUN 通用公共许可证)的方式来发行传播，也就是说这个软件允许任何人以任何形式进行修改和传播。

随着网络的日益盛行，越来越多的技术高超的程序员加入到 Linux 操作系统的开发与完

善中来。在这个过程中，无数的富有个性和开创性的程序员在没有计较任何酬劳的前提下，完全自发地加入到开发行列中来。一旦一个程序员完成了其中的部分程序，他便会立即将这个程序发表，并免费将它发给任何一个需要的人，而其他的一些程序员研究它后将会对它修正和改良，然后将它发表。这样的过程周而复始，因此 Linux 操作系统的改进速度是最快的，同时它的稳定性也是非常高的。所以，Linux 操作系统并非仅由 Linus 一人开发，而是由全世界很多个程序员共同开发，当然 Linus 为内核定了“调子”。这种集市型的开发模式促成了 Linux 操作系统的繁荣。可以说，Linux 操作系统完全是一个热情、自由、开放的网络产物。

3.1.3　Linux 操作系统的发展

Linux 操作系统具有良好的兼容性和可移植性。大约在 1.3 版本之后，Linux 操作系统开始向其他硬件平台上移植，包括号称最快的 CPU——Digital Alpha。所以不要总把 Linux 操作系统与低档硬件平台联系到一起，Linux 操作系统只是将硬件的性能充分发挥出来而已。Linux 操作系统必将从低端应用横扫到高端应用!

为了使 Linux 操作系统变得容易使用，Linux 操作系统也有了许多发布版本，发布版本实际上就是一整套完整的程序组合。现在已经有许多不同的 Linux 操作系统发行版和各自的版本号，为了不产生混淆，先解释一些常提到的术语。当人们提到的“Linux”时，一般是指“Real Linux”，即内核，是所有 UNIX 操作系统的“心脏”。但光有 Linux 并不能成为一个可用的操作系统，还需要许多软件包、编译器、程序库文件、Xwindow 系统等。因为组合方式不同、面向用户对象不同，所以就有了许多不同的 Linux 操作系统发行版。

越来越多的公司在 Linux 操作系统上开发商业软件或把其他 UNIX 操作系统平台的软件移植到 Linux 操作系统上来。如今很多 IT 业界的“大腕”，如 IBM、Intel、Oracle、Infomix、Sysbase、Corel、Netscape、CA、Novell 等公司都宣布支持 Linux 操作系统。商家的加盟弥补了纯自由软件的不足和发展障碍，Linux 操作系统迅速普及到广大计算机爱好者范围，并且进入商业应用，成为打破某些公司形成垄断的希望所在。

Linux 操作系统是爱好者们通过互联网协同开发出来的，当然它的网络功能十分强大。比如可以通过 FTP、NFS 等来安装 Linux 操作系统，用它来做网关等。随着 Linux 操作系统的发展，衍生出来的应用恐怕出乎 Linus 本人最初的预料，如有人用它来做路由器、有人来做嵌入式系统、有人来做实时性系统。常有新手问 Linux 操作系统能做什么？其实它不像那些中看不中用的操作系统，不在于你用它能干什么，而在于你想干什么。

Linux 操作系统是一个在 PC 上运行的 UNIX 操作系统。Linux 操作系统具有最新 UNIX 操作系统的全部功能，包括真正的多任务、虚拟存储、共享库函数、即时负载、优越的存储管理和 TCP/IP、UUCP 网络工具等。Linux 操作系统及其发展均符合 Posix 标准，其内核支持 Ethernet、PPP、SLIP、NFS、AX.25、IPX/SPX(Novell)、NCP(Novell)等。系统应用包括 Tellnet、Rlogin、FTP、Mail、gopher、talk、term、news(tin、trn、nn)等全套 UNIX 操作系统工具包。X 图形库，包括 xterm、fvwm、xxgdb、mosaic、xv、gs、xman 等全部 X-Win 应用工具。商业软件有 Motif、WordPerfect。中文工具已有 Cxterm、celvis、cemasc、cless、hztty、cytalk、ctalk、cmail 等，可以处理 GB、BIG5、HZ 文件。此外还有 DOS 模拟软件，可以运行 DOS/Windows 操作系统下的软件。

在开始的时候，Linux 操作系统只是个人狂热爱好的一种产物。但是现在，Linux 操作系统已经成为一种受到广泛关注和支持的操作系统。和其他的商用 UNIX 操作系统相比，作为自由软件的 Linux 操作系统具有低成本、安全性高、更加可信赖的优势。直到今天，Linux 操作系统已经成为一个功能完善的主流网络操作系统。

3.2　Linux 操作系统常用的 shell 命令

3.2.1　基本命令

1）立即关机并重启动，执行如下命令：

shutdown　-r　now

或者 reboot

2）立即关机，执行如下命令：

shutdown　-h　now

或者 poweroff

3）等待 2min 关机并重启动，执行如下命令：

shutdown　-r　2

4）等待 2min 关机，执行如下命令：

shutdown　-h　2

5）使用当前用户的历史命令，执行如下命令：

history

将会显示使用过的每条命令及其序号，可利用序号重复执行该命令。例如输入！1 并回车，将会重复执行第 1 条历史命令。也可用上下光标键调出某条历史命令，然后按回车键重复执行。还可用上下光标键调出某条历史命令，修改后按回车键执行。

6）清除当前用户的历史命令，执行如下命令：

history　-c

此时用向上光标键将会调不出任何历史命令。

7）命令提示键“TAB”：输入命令开头一个或几个字母，然后按 1 次“TAB”键，系统会自动补全能够识别的部分；再按 1 次“TAB”键，系统显示出符合条件的所有命令供用户选择。

例如输入 group 后按两次“TAB”键，将会显示以 group 开头的所有命令。

8）显示内核版本号，执行如下命令：

uname　-r

注意：内核版本号不同于软件发行版本号。例如，RHEL 5.4 的内核版本号是 2.6.18-164.el5，软件发行版本号是 5.4。

9）清除屏幕，执行如下命令：

clear

10）显示操作系统时钟，执行如下命令：

date

11）加载光盘到/media，执行如下命令：

mount　/dev/cdrom　/media

12）卸载光盘，执行如下命令：

umount　/dev/cdrom

或者

umount　/media

注意：不要在/media 或其子目录中执行此命令，否则将会出现“设备忙错误”。

13）查看存储设备，执行如下命令：

fdisk-1

14）加载 U 盘到/media，执行如下命令：

mount　/dev/sdb1　/media

15）卸载 U 盘，执行如下命令：

umount　/dev/sdb1

或者

umount　/media

注意：不要在/media 或其子目录中执行此命令，否则将会出现“设备忙错误”。

16）中断 shell 命令，执行如下命令：

Ctrl + C

3.2.2　文件目录操作命令

1）显示当前的绝对路径，执行如下命令：

pwd

2）改变当前目录，执行如下命令：

cd　/etc/yum

将会把当前目录改为/etc/yum。

3）回到当前目录的父目录，执行如下命令：

cd　..

4）创建目录，执行如下命令：

mkdir　/usr/tigger

5）删除目录，执行如下命令：

rmdir　/usr/tigger

注意：使用 rmdir 命令时，待删除的目录必须为空。

6）列出目录中的内容，执行如下命令：

ls　/

7）列出目录中的所有内容(包括隐藏文件或称为点文件)，执行如下命令：

ls　/root　-a

将会看到以“.”开头的文件名，它们称为点文件。若用命令“ls　/root”命令是看不到它们的。

8）用长格式列出目录中的内容，执行如下命令：

ls /boot -l

注意：在Linux操作系统中，若某命令有几个开关，可将这几个开关合并在一起。例如，命令“ls -a -l”与命令“ls -al”或者“ls -la”作用相同。

9）创建空文件，执行如下命令：

touch /a. dat

10）复制文件，执行如下命令：

cp /etc/host. conf /root

将会把目录/etc中的文件host. conf复制到目录/root中，文件名不变。

11）复制整个子目录(不改变目录名)，执行如下命令：

cp -r /usr/include /root

将会把整个子目录/usr/include(不改变目录名)复制到目录/root中。

12）复制整个子目录(改变目录名)，执行如下命令：

cp -r /usr/include /root/include2

将会把整个子目录/usr/include复制到目录/root中，并将目录名从include改为include2。

13）移动文件或给文件改名，执行如下命令：

给文件改名：

mv /root/host. conf /root/myfile

移动文件：

mv /root/myfile /

移动文件同时改名：

mv /myfile /root/myfile2

14）删除文件，执行如下命令：

rm /root/myfile2

按“y”键确认。

rm -f /a. dat

不需确认。

15）删除非空目录，执行如下命令：

mkdir /root/mysub /root/mysub/new

rmdir /root/mysub

系统提示目录非空。

rm -rf /root/mysub

系统无错误提示。

ls /root

将看到目录/root中已经没有mysub目录。

16）分屏显示文件内容，执行如下命令：

more /etc/services

按空格键显示下一屏，按“q”键返回命令行状态。

注意：more作为管道命令时，可与其他一些命令结合，例如：

ls　/etc　| more

history　| more

17）显示文件内容，执行如下命令：

cat　/etc/services

18）合并文件，执行如下命令：

cat　/etc/resolv. conf　/etc/yum. conf　>/b. dat

执行如下命令进行验证：

ls　-l　/b. dat

显示该文件长度为 814B。

也可用两条命令实现同样的功能：

cat　/etc/resolv. conf　>/c. dat

此时该文件长度为 26B。

cat　/etc/yum. conf　>>/c. dat

此时该文件长度为 814B。

注意：>和>>是重定向符号，若重定向的文件已经存在，则使用>时将用新内容覆盖原来的内容，而使用>>时将用新内容添加到原来内容的后面。

3. 2. 3　vi 编辑器

创建或修改某一文本文件，执行如下命令：

vi　/b. dat

vi 编辑器有两种模式：命令模式和编辑模式。

vi 启动后进入的是命令模式，在命令模式中按“i”键就可以进入编辑模式。在编辑模式中按“Esc”键就可以返回到命令模式。

按 i 键后开始编辑。编辑完成后，按“Esc”键返回到命令模式，输入“：wq”后按回车键保存文件后退出；或者输入“：q!”后按回车键不存盘退出。

若要删除光标所在行，则先返回到命令模式，再按两次“d”键。若要删除从光标所在行开始向下的若干行，例如 5 行，则先返回到命令模式，按“5”键，再按两次“d”键。删除的内容同时进入 vi 缓冲区。

若要将 vi 缓冲区的内容粘贴到当前位置的后面，则先返回到命令模式，再按“p”键。

若要撤销最近一次的操作，则先返回到命令模式，再按“u”键。重复按“u”键可以撤销最近的多次操作。

若要将光标所在行复制到 vi 缓冲区，则先返回到命令模式，再按两次“y”键。若要将从光标所在行开始向下的若干行（例如 5 行）复制到 vi 缓冲区，则先返回到命令模式，按“5”键，再按两次“y”键。

若要从当前位置开始向下查找某一字符串，例如 HOSTNAME，则先返回到命令模式，再输入/HOSTNAME 后按回车键。若要继续向下查找，则再输入“/”后按回车键。

vi 在编辑某一个文件时，会生成一个临时文件，这个文件以“. ”开头并以“. swp”结尾。正常退出该文件自动删除，如果意外退出例如忽然断电，该文件不会删除。此时手动删除该文件即可。

：set nu 显示行号

：setnonu 取消行号

3.2.4　软件包安装命令

1）查看所有已安装的软件包，执行如下命令：

rpm　-qa　| more

2）查看已安装的名称中包含某个字符串的所有软件包，例如执行如下命令：

rpm　-qa　| grep　net

3）验证所有已安装的软件包，执行如下命令：

rpm　-Va

注意：该命令会列出所有自从包安装后系统和用户做过修改的文件。

4）查看已安装的某个软件包的用途，执行如下命令：

rpm　-qi　net-tools-1.60-37.EL4.8

5）查看系统中某个文件属于哪个软件包，执行如下命令：

rpm　-qf　/sbin/ifconfig

结果应显示该文件属于 net-tools-1.60-102.e16.i686。

6）安装某个软件包，执行如下命令：

rpm　-ivh　* * * * * * * * * * * *.rpm

注意：-v 为显示信息选项，-h 为显示进程选项。

第 2 篇

基础实践部分

第 4 章　集群搭建

4. 1　CentOS 操作系统的安装

4. 1. 1　实验目的

1）熟悉 Linux 操作系统安装过程。

2）了解安装配置信息。

4. 1. 2　实验设备

1）硬件：PC。

2）软件：CentOS 5. X。

4. 1. 3　实验内容

在 PC 上安装 CentOS 操作系统。

4. 1. 4　实验步骤

1）系统版本：CentOS 5. 5，将镜像刻成光盘，设置 BIOS 将 CD-ROM 设置为第一启动。安装启动界面如图 4-1 所示。

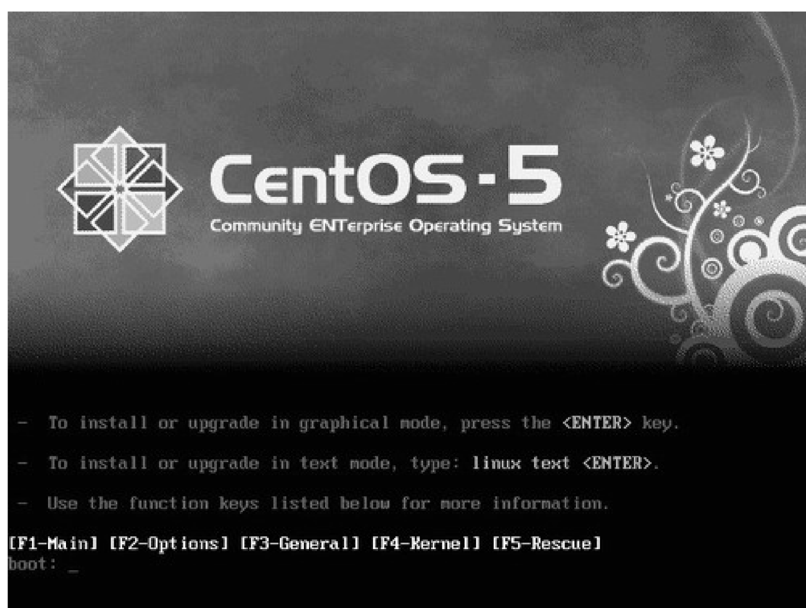


图 4-1　安装启动界面

2）通过提示，按“ENTER”键进入图形安装模式，如图 4-2 所示。

```
md: bitmap version 4.39
TCP bic registered
Initializing IPsec netlink socket
NET: Registered protocol family 1
NET: Registered protocol family 17
Using IPI No-Shortcut mode
Time: tsc clocksource has been installed.
ACPI: (supports S0 S1 S4 S5)
Initalizing network drop monitor service
Freeing unused kernel memory: 228k freed
Write protecting the kernel read-only data: 409k

Greetings.
anaconda installer init version 11.1.2.209 starting
mounting /proc filesystem... done
creating /dev filesystem... done
mounting /dev/pts (unix98 pty) filesystem... done
mounting /sys filesystem... done
input: AT Translated Set 2 keyboard as /class/input/input0
input: ImPS/2 Generic Wheel Mouse as /class/input/input1
trying to remount root filesystem read write... done
mounting /tmp as ramfs... done
running install...
running /sbin/loader
```

图 4-2　信息检测界面

信息检测，开始安装，如图 4-3 所示。

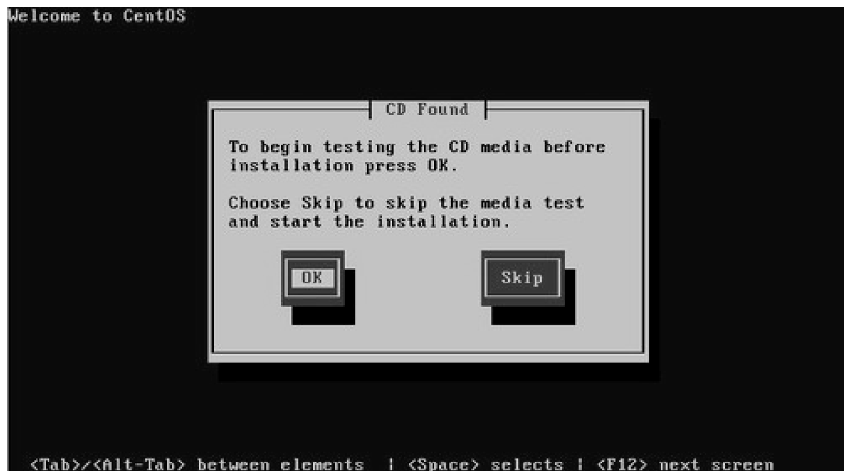


图 4-3　选择图形界面安装

3）检测安装媒介，比如我们是用光盘安装，即为检测光盘安装数据的完整性，我们可通过按“TAB”键选择“OK”或者是“Skip”，这里选的是“Skip”，即跳过。

安装程序的第一个图形界面如图 4-4 所示。

4）选择安装时的语言，这里选择简体中文，如图 4-5 所示。

5）键盘语言，默认即可，如图 4-6 所示。

6）这是一块新硬盘，没有初始化，也没有分区，所以提示进行初始化，如图 4-7 所示。

7）硬盘分区方案，系统默认是按照系统定义好的方式进行划分，但是在这里，选择手动，如图 4-8 所示。

8）最简单的就是创建两个分区，一个根一个交换分区，如图 4-9 和图 4-10 所示。



图 4-4 安装程序的第一个图形界面



图 4-5 选择语言界面



图 4-6 选择键盘语言界面



图 4-7 清除数据界面



图 4-8 选择分区界面



图 4-9 配置根分区界面



图 4-10　配置交换分区界面

9）在 CentOS 中，引导程序采用的是 GRUB，如图 4-11 所示。



图 4-11　GRUB 引导程序

10）配置网络连接方式及主机名，如图 4-12 所示。



图 4-12　配置网络连接方式及主机名

11）选择所在区域，如图 4-13 所示。



图 4-13　选择所在区域

12）为根用户 root（根）设置一个密码，在这里 root 是系统管理员拥有至高的权利，所以

必须设置一个强有力的密码，如图 4-14 所示。



图 4-14　设置根用户密码

13）选择桌面环境，常见的有 Gnome、KDE，这里选择 Gnome，如图 4-15 所示。



图 4-15　选择桌面环境

检查安装时的依赖关系如图 4-16 所示。



图 4-16　检查安装时的依赖关系

14）点击“下一步”按钮正式安装，如图 4-17 所示。



图 4-17　开始正式安装

接下来的步骤如图 4-18 ~ 图 4-24 所示。



图 4-18 格式化文件系统



图 4-19 开始安装进程



图 4-20　安装中



图 4-21　安装完成

```
Probing for video card:   VMware SVGA II Adapter
Attempting to start native X server
Waiting for X server to start...log located in /tmp/ramfs/X.log
1...2...3...4...5... X server started successfully.
Starting graphical installation...
sending termination signals...done
sending kill signals...done
disabling swap...
        /tmp/sda2
unmounting filesystems...
        /mnt/runtime done
        disabling /dev/loop0
        /proc/bus/usb done
        /proc done
        /dev/pts done
        /sys done
        /tmp/ramfs done
        /selinux done
        /mnt/sysimage/sys done
        /mnt/sysimage/proc done
        /mnt/sysimage/selinux done
        /mnt/sysimage/dev done
        /mnt/sysimage done
rebooting system
_
```

图 4-22　重启中

```
  Booting 'CentOS (2.6.18-194.el5)'

root (hd0,0)
 Filesystem type is ext2fs, partition type 0x83
kernel /boot/vmlinuz-2.6.18-194.el5 ro root=LABEL=/ rhgb quiet
   [Linux-bzImage, setup=0x1e00, size=0x1c7f54]
initrd /boot/initrd-2.6.18-194.el5.img
   [Linux-initrd @ 0x1d05f000, 0x200c0d bytes]

Memory for crash kernel (0x0 to 0x0) notwithin permissible range
_
```

图 4-23　装载内核



图 4-24 启动信息

15）安装完成后，还需要一些简单的配置，根据提示进行即可，如图 4-25 所示。



图 4-25 进入基本配置

16）这里的防火墙是指 iptables，一般设置为禁用，需要的时候自行启动，如图 4-26

所示。



图 4-26　配置防火墙

17）SELinux 选项是更安全的行为控制，设置为强制，如图 4-27 所示。



图 4-27　配置 SELinux 选项

18）系统时间一般不用设置，如图 4-28 所示。



图 4-28 配置系统时间

19）我们知道，root 拥有最高的权利，所以让 root 登录进去不太安全，所以设置一个普通用户，需要的时候切换到 root，如图 4-29 所示。



图 4-29 设置普通用户

20）声卡测试没什么用，服务器上不需要声卡，如图 4-30 所示。



图 4-30　声卡测试

21）安装一些其他的软件，这里就不需要了，需要的时候自行安装，如图 4-31 所示。



图 4-31　是否安装其他软件

22）系统登录，用刚才创建好的用户名和密码登录，如图 4-32 所示。



图 4-32　系统登录界面

23）至此 CentOS 软件安装成功，如图 4-33 所示。



图 4-33　安装成功界面

24）修改三台机器的 IP 地址，分别是 192.168.0.101、192.168.0.102 和 192.168.0.103，子网掩码都改成 255.255.255.0。

4.2　集群搭建

4.2.1　实验目的

1）熟悉 Hadoop 集群搭建过程。

2）掌握 Hadoop 核心程序配置方法。

4.2.2　实验设备

1）硬件：云计算实验一体机、PC。

2）软件：SecureCRT、FileZilla、JDK 安装包。

4.2.3　实验内容

1）把安装程序上传到集群并且安装 JDK。

2）无密钥 SSH 登录。

3）核心程序配置。

4.2.4　实验步骤

1. 针对集群中的所有机器进行配置

1）打开连接工具 SecureCRT，如图 4-34 ~ 图 4-36 所示。

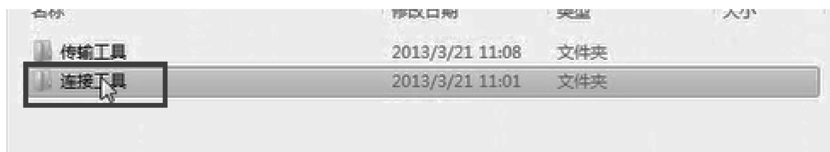


图 4-34　连接工具



图 4-35　SecureCRT 工具

2）打开传输工具 FileZilla，如图 4-37 ~ 图 4-39 所示。

3）连接三台主机，点击第三个按钮，连接主机，如图 4-40 所示。

4）选择要连接的主机 IP，如果之前没有连接过，可以点击快速连接，输入对方 IP 地址和用户名进行连接，如图 4-41 所示。

5）输入用户名和密码，先用 root 用户登录，如图 4-42 所示。

6）三台主机登录成功，如图 4-43 所示。

Microsoft.VC90.MFCLOC.manifest	2011/3/9 2:02	MANIFEST 文件	6 KB
migrate	2011/3/9 2:02	应用程序	1,607 KB
msvcp80.dll	2011/3/9 2:02	应用程序扩展	536 KB
msvcp90.dll	2011/3/9 2:02	应用程序扩展	557 KB
msvcr80.dll	2011/3/9 2:02	应用程序扩展	612 KB
msvcr90.dll	2011/3/9 2:02	应用程序扩展	638 KB
Rlogin.dll	2011/3/9 2:02	应用程序扩展	186 KB
SecureCRT	2011/3/9 2:02	编译的 HTML 帮...	896 KB
SecureCRT	2011/3/9 2:02	应用程序	2,782 KB
SecureCRT_EULA	2011/3/9 2:02	文本文档	6 KB
SecureCRT_HISTORY	2011/3/9 2:02	文本文档	17 KB
SecureCRT_Order	2011/3/9 2:02	文本文档	5 KB
SecureCRT_README	2011/3/9 2:02	文本文档	13 KB
Serial.dll	2011/3/9 2:02	应用程序扩展	200 KB

图 4-36　SecureCRT 图标

名称	修改日期	类型	大小
传输工具	2013/3/21 11:08	文件夹	
连接工具	2013/3/21 11:01	文件夹	

图 4-37　传输工具

名称	修改日期	类型	大小
FileZilla FTP Client	2013/3/21 11:00	文件夹	
FileZilla FTP Client	2013/3/21 10:50	360压缩 RAR 文件	5,159 KB
FileZilla_3.6.0.2_win32-setup	2013/3/21 11:08	应用程序	4,593 KB

图 4-38　FileZilla 工具

名称	修改日期	类型	大小
docs	2013/3/21 11:00	文件夹	
locales	2011/12/19 13:31	文件夹	
resources	2013/3/21 11:00	文件夹	
AUTHORS	2010/11/8 6:03	媒体文件	3 KB
filezilla	2010/11/21 22:54	应用程序	7,443 KB
fzputtygen	2010/11/21 22:54	应用程序	127 KB
fzsftp	2010/11/21 22:54	应用程序	335 KB
fzshellext.dll	2010/11/21 22:54	应用程序扩展	92 KB
fzshellext_64.dll	2010/1/2 22:42	应用程序扩展	96 KB
GPL	2007/11/23 19:34	360 se HTML Do...	16 KB
mingwm10.dll	2010/1/2 22:42	应用程序扩展	18 KB
NEWS	2010/11/21 21:41	媒体文件	48 KB
uninstall	2011/12/19 13:31	应用程序	62 KB

图 4-39　FileZilla 图标

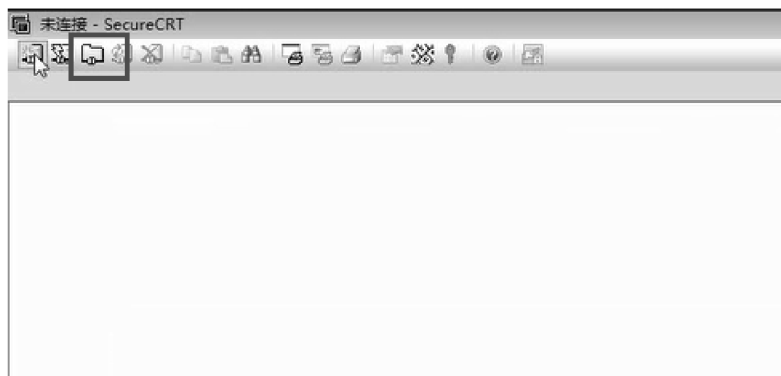


图 4-40　SecureCRT 工具界面

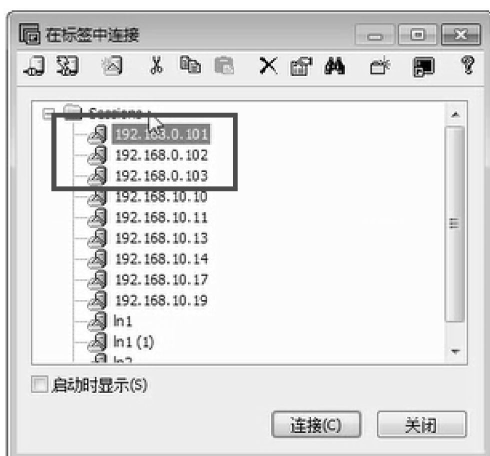


图 4-41　快速连接对话框



图 4-42　登录对话框

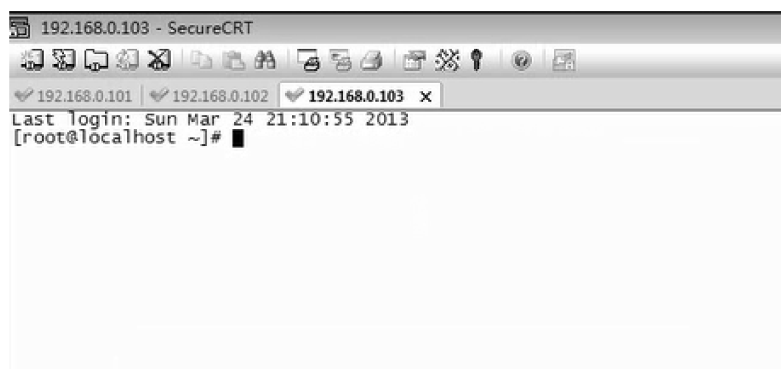


图 4-43　登录成功

7）新增用户 hadoop。使用 root 账户登录系统，添加用户：useradd hadoop；密码修改：passwd hadoop，提示输入新密码和重复输入新密码，然后退出 root 用户，如图 4-44 ~ 图4-46 所示。



图 4-44　新增用户 hadoop

```
[root@localhost ~]# passwd hadoop
Changing password for user hadoop.
New UNIX password:
BAD PASSWORD: it is based on a dictionary word
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@localhost ~]#
```

图 4-45　修改密码

```
passwd: all authentication tokens up
[root@localhost ~]# exit
```

图 4-46　退出 root 用户

8）打开会话选项，如图 4-47 所示。



图 4-47　打开会话选项

9）使用新建用户 hadoop 登录，如图 4-48 所示。

10）在“用户名”里输入 hadoop，如图 4-49 所示。

11）重新连接主机，如图 4-50 所示。

12）使用 hadoop 用户登录三台主机，并保存密码，如图 4-51 所示。



图 4-48　设置登录用户



图 4-49　新建用户 hadoop



图 4-50　重新连接主机



图 4-51　用 hadoop 用户登录并保存密码

13）在会话选项中更改外观，如图 4-52 所示。



图 4-52　更改外观

14）将字符编码改成 UTF-S，如图 4-53 所示。

15）Linux 操作系统配置。用 hadoop 用户登录：

```
su root
```

新建/hadoop 目录，如图 4-54 所示：

```
mkdir /hadoop
```



图 4-53　更改字符编码

```
[hadoop@localhost /]$ su root
口令:
[root@localhost /]#
```

图 4-54　新建/hadoop 目录

使用 ll /命令查看是否创建成功，如图 4-55 所示。

```
[root@localhost /]# mkdir /hadoop
[root@localhost /]# ll /
总计 154
drwxr-xr-x   2 root root  4096 03-22 01:50 bin
drwxr-xr-x   4 root root  1024 03-20 01:18 boot
drwxr-xr-x  13 root root  3980 03-24 21:06 dev
drwxr-xr-x 101 root root 12288 03-24 21:10 etc
drwxr-xr-x   2 root root  4096 03-24 21:22 hadoop
drwxr-xr-x   4 root root  4096 03-24 21:10 home
drwxr-xr-x  11 root root  4096 03-22 01:50 lib
drwxr-xr-x   7 root root  4096 03-22 01:50 lib64
drwx------   2 root root 16384 03-20 01:16 lost+found
drwxr-xr-x   3 root root  4096 03-24 21:10 media
drwxr-xr-x   2 root root     0 03-24 21:06 misc
drwxr-xr-x   2 root root  4096 2010-01-27 mnt
drwxr-xr-x   2 root root     0 03-24 21:06 net
drwxr-xr-x   2 root root  4096 2010-01-27 opt
dr-xr-xr-x 168 root root     0 03-24 21:05 proc
drwxr-x---  16 root root  4096 03-24 21:19 root
drwxr-xr-x   2 root root 12288 03-22 01:50 sbin
drwxr-xr-x   4 root root     0 03-24 21:05 selinux
drwxr-xr-x   2 root root  4096 2010-01-27 srv
drwxr-xr-x  11 root root     0 03-24 21:05 sys
drwxrwxrwt  11 root root  4096 03-24 21:10 tmp
drwxr-xr-x  15 root root  4096 03-20 01:17 usr
drwxr-xr-x  21 root root  4096 03-20 01:20 var
[root@localhost /]#
```

图 4-55　使用 ll/命令查看是否创建成功

16）然后授权给 hadoop 用户，如图 4-56 所示：

chown -R hadoop：hadoop /hadoop

```
[root@localhost /]# chown -R hadoop:hadoop /hadoop/
[root@localhost /]# ll /
drwxr-xr-x   2 root   root     4096 03-22 01:50 bin
drwxr-xr-x   4 root   root     1024 03-20 01:18 boot
drwxr-xr-x  13 root   root     3980 03-24 21:06 dev
drwxr-xr-x   2 hadoop hadoop   4096 03-24 21:22 hadoop
drwxr-xr-x  11 root   root     4096 03-22 01:50 lib
drwxr-xr-x   7 root   root     4096 03-22 01:50 lib64
drwx------   2 root   root    16384 03-20 01:16 lost+found
drwxr-xr-x   3 root   root     4096 03-24 21:10 media
drwxr-xr-x   2 root   root        0 03-24 21:06 misc
drwxr-xr-x   2 root   root     4096 2010-01-27 mnt
drwxr-xr-x   2 root   root        0 03-24 21:06 net
drwxr-xr-x   2 root   root     4096 2010-01-27 opt
dr-xr-xr-x 168 root   root        0 03-24 21:05 proc
drwxr-x---  16 root   root     4096 03-24 21:19 root
drwxr-xr-x   2 root   root    12288 03-22 01:50 sbin
drwxr-xr-x   4 root   root        0 03-24 21:05 selinux
drwxr-xr-x   2 root   root     4096 2010-01-27 srv
drwxr-xr-x  11 root   root        0 03-24 21:05 sys
drwxrwxrwt  11 root   root     4096 03-24 21:10 tmp
drwxr-xr-x  15 root   root     4096 03-20 01:17 usr
drwxr-xr-x  21 root   root     4096 03-20 01:20 var
[root@localhost /]#
```

图 4-56　授权给 hadoop 用户

17）修改主机名，暂时性修改主机名(重启就会消失)，如图 4-57 所示：

hostname $ name(master/slave1/slave2)

备注：其中$ name 的值为要修改的主机名。

```
[root@localhost /]# hostname
localhost.localdomain
[root@localhost /]# hostname master
[root@localhost /]# hostname
master
[root@localhost /]#
```

图 4-57　修改主机名

18）从文件中更改主机名，如图 4-58 所示：

vim /etc/sysconfig/network

把文件里 HOSTNAME 的值修改为刚才的$ name(master/slave1/slave2)，如图 4-59 所示。

```
master
[root@localhost /]# vim /etc/sysconfig/network
```

图 4-58　从文件中更改主机名

```
NETWORKING=yes
NETWORKING_IPV6=no
HOSTNAME=localhost.localdomain
```

```
NETWORKING=yes
NETWORKING_IPV6=no
HOSTNAME=master
```

图 4-59　把文件里 HOSTNAME 的值修改为刚才的$ name(master/slave1/slave2)

19）查看修改是否成功：

hostname

是否输出为修改的$ name。

20）关闭防火墙，如图 4-60 所示：

/etc/init. d/iptables stop

备注：关闭当前防火墙。

/sbin/service iptables stop

备注：关闭防火墙服务。

```
[root@localhost /]# /etc/init.d/iptables stop
[root@localhost /]# /sbin/service iptables stop
[root@localhost /]# █
```

图 4-60　关闭防火墙

21）验证时关闭防火墙

/etc/init. d/iptables status

验证防火墙服务是否停止，如图 4-61 所示：

/sbin/service iptables status

22）退出 root 用户，如图 4-62 所示。

```
[root@localhost /]# /etc/init.d/iptables status
防火墙已停
[root@localhost /]# /sbin/service iptables status
防火墙已停
[root@localhost /]# █
```

图 4-61　验证防火墙服务是否停止

```
[root@localhost /]# exit
exit
[hadoop@localhost /]$
```

图 4-62　退出 root 用户

23）在部署前，上面所说的 Windows Server 需要简单地配置一下，首先修改 C：\ Windows \ System32 \ drivers \ etc \ hosts 文件，把 master 和 slaves 的 IP 和主机名称写在这里，一台一行，示例如下：

192. 168. 0. 101master

192. 168. 0. 102slave1

192. 168. 0. 103slave2

这样确保这台 Windows Server 可以正确识别这些主机名称，如图 4-63 所示。

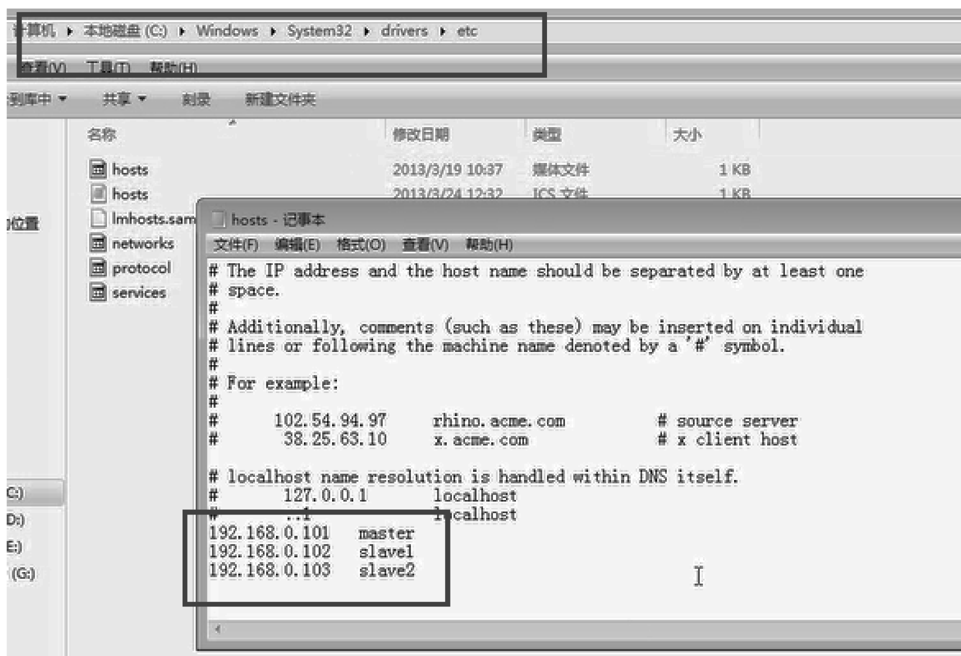


图 4-63　配置 Windows Server

24）安装 JDK（版本必须是 1. 6 以上），配置 Windows Server 如图 4-64 所示。

25）连接 master（主机），传输 jdk 安装包，如图 4-65 所示。



图 4-64　配置 Windows Server

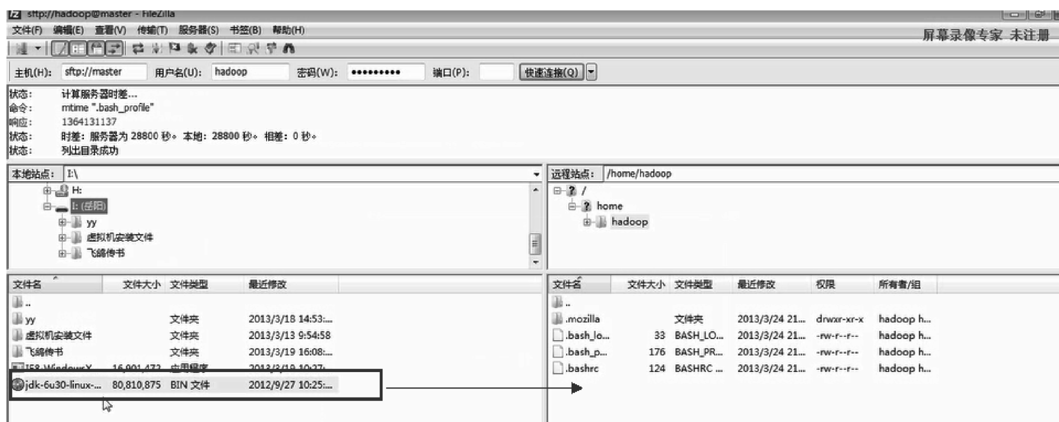


图 4-65　传输 jdk 安装包

26）为了方便看出文件的权限，在会话选项中设置终端为“Linux”，如图 4-66 和图 4-67所示。



图 4-66　设置终端



图 4-67　设置终端为“Linux”

27）外观中设置当前颜色方案为“Tradtional”，如图 4-68 和图 4-69 所示。



图 4-68　设置当前颜色方案

28）看一下是否复制成功，cd ~/先到根文件目录下，ll 查看文件，如图 4-70 所示。

29）进入到 root 用户，su root，移动文件到/usr/local/下安装，mv jdk-6u30-linux-i586-rpm. bin /usr/local/，如图 4-71 所示。



图 4-69 设置当前颜色方案为“Tradtional”

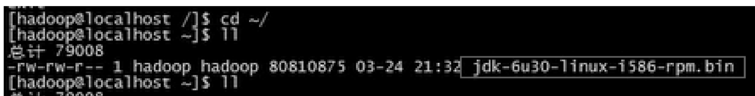


图 4-70 查看是否复制成功



图 4-71 进入到 root 用户

30）看一下是否移动成功，cd /usr/local，ll 查看文件，如图 4-72 所示。

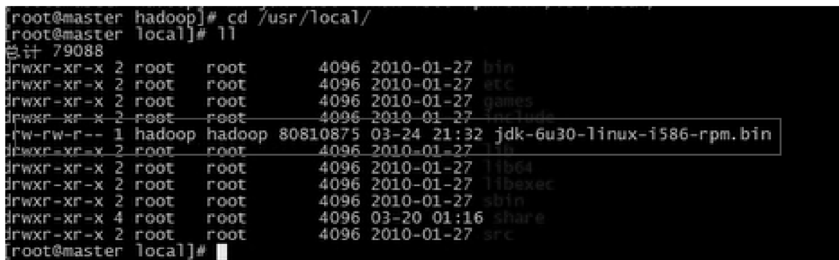


图 4-72 看一下是否移动成功

31）授权给这个文件可执行权限，chmod +x jdk-6u30-linux-i586-rpm. bin，ll，如图 4-73 所示。

32）安装 JDK， ./ jdk-6u30-linux-i586-rpm. bin，如图 4-74 所示。

等待…，如图 4-75 所示。

```
[root@master local]# chmod +x jdk-6u30-linux-i586-rpm.bin
[root@master local]# ll
总计 79088
drwxr-xr-x 2 root   root        4096 2010-01-27 bin
drwxr-xr-x 2 root   root        4096 2010-01-27 etc
drwxr-xr-x 2 root   root        4096 2010-01-27 games
drwxr-xr-x 2 root   root        4096 2010-01-27 include
-rwxrwxr-x 1 hadoop hadoop 80810875 03-24 21:32 jdk-6u30-linux-i586-rpm.bin
drwxr-xr-x 2 root   root        4096 2010-01-27 lib
drwxr-xr-x 2 root   root        4096 2010-01-27 lib64
drwxr-xr-x 2 root   root        4096 2010-01-27 libexec
drwxr-xr-x 2 root   root        4096 2010-01-27 sbin
drwxr-xr-x 4 root   root        4096 03-20 01:16 share
drwxr-xr-x 2 root   root        4096 2010-01-27 src
[root@master local]#
```

图 4-73　授权给这个文件可执行权限

```
drwxr-xr-x 2 root   root        4096 2010-01-27 src
[root@master local]# ./jdk-6u30-linux-i586-rpm.bin
```

图 4-74　安装 JDK

```
Unpacking...
Checksumming...
Extracting...
UnZipSFX 5.50 of 17 February 2002, by Info-ZIP (Zip-Bugs@lists.wku.edu).
  inflating: jdk-6u30-linux-i586.rpm
  inflating: sun-javadb-common-10.6.2-1.1.i386.rpm
  inflating: sun-javadb-core-10.6.2-1.1.i386.rpm
  inflating: sun-javadb-client-10.6.2-1.1.i386.rpm
  inflating: sun-javadb-demo-10.6.2-1.1.i386.rpm
  inflating: sun-javadb-docs-10.6.2-1.1.i386.rpm
  inflating: sun-javadb-javadoc-10.6.2-1.1.i386.rpm
Preparing...                ########################################### [100%]
   1:jdk                    ########################################### [100%]
```

图 4-75　等待…

33）安装完成，如图 4-76 所示。

```
For more information on what data Registration collects and
how it is managed and used, see:
http://java.sun.com/javase/registration/JDKRegistrationPrivacy.html

Press Enter to continue.....
^[[6~

Done.
[root@master local]#
```

图 4-76　安装完成

34）退出 root，exit，如图 4-77 所示。

35）安装成功后，设置系统环境变量，编辑 bash-profile 文件，vim ~/. bash_profile，如图 4-78 所示。

```
[root@master local]# exit
exit
```

图 4-77　退出 root

```
[hadoop@localhost ~]$ vim ~/.bash_profile
```

图 4-78　编辑 bash_profile 文件

36）在文件的最后面加入以下行：

#set java environment

JAVA_HOME =/usr/java/jdk1. 6. 0_30

PATH =$JAVA_HOME/bin：$PATH

CLASSPATH =$CLASSPATH：.：$JAVA_HOME/lib

export JAVA_HOME CLASSPATH PATH

37）最后，退出 hadoop 用户，重新登录 hadoop，检验 java 是否安装成功，如图 4-79 所示：

java-version

如果输出 java version " 1.6.0_30" 等一系列信息，则表示安装成功，如图 4-80 所示。

38）修改 ulimit 数量，进入 root 用户，vim /etc/security/limits. conf，如图 4-81 所示。

```
~
".bash_profile" 17L, 326C 已写入
[hadoop@localhost ~]$ ll
总计 0
[hadoop@localhost ~]$ exit
```

图 4-79　检验 java 是否安装成功

```
Last login: Sun Mar 24 21:19:40 2013 from 192.168.0.99
[hadoop@master ~]$ java -version
java version "1.6.0_30"
Java(TM) SE Runtime Environment (build 1.6.0_30-b12)
Java HotSpot(TM) Server VM (build 20.5-b03, mixed mode)
[hadoop@master ~]$
```

图 4-80　安装成功

```
[hadoop@master ~]$ su root
口令:
[root@master hadoop]# vim /etc/security/limits.conf
```

图 4-81　进入 root 用户

39）修改其内容，在文件最后加入（数值也可以自己定义）：

* soft nofile 10240

* hard nofile 10240

40）vim /etc/profile，修改其内容，在最后加入图 4-82 和图 4-83 所示内容。

```
[root@master hadoop]# vim /etc/profile
```

图 4-82　修改内容

```
unset i
unset pathmunge
ulimit -n 10240
"/etc/profile" 59L, 1045C 已写入
[root@master hadoop]# exi
```

图 4-83　加入内容

41）退出 root 用户，然后重新登录 hadoop，用 ulimit-n 命令验证，如图 4-84 所示。

以上是针对集群中的所有机器进行的配置。如若不需要做时间同步服务器，则至少应该保证所有集群的时间相差无几。

2. 无密钥 SSH 登录

1）首先修改/etc/hosts 文件，所有的机器上都需要修改。

进入 root 用户，如图 4-85 所示：

```
[hadoop@master ~]$ ulimit -n
10240
```

图 4-84　用 ulimit -n 命令验证

```
[hadoop@master .ssh]$ vim /etc/host
```

图 4-85　进入 root 用户

su root

vim /etc/hosts

添加集群里所有的机器 IP 和主机名称，如图 4-86 所示。

osts 文件，所有的机器上都需要修改，vim /etc/hosts（三台机器都需要操作）。

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1              localhost.localdomain localhost
::1             localhost6.localdomain6 localhost6
192.168.0.101   master
192.168.0.102   slave1
192.168.0.103   slave2
```

图 4-86　添加集群里所有的机器 IP 和主机名称

2）切换到 master，退出 root 用户。

使用 hadoop 用户执行，如图 4-87 所示：

ssh-keygen -t dsa

一路回车下去(三台机器都需要操作)。此时会在 ~/. ssh 目录下生成一对密钥，一个公钥 id_dsa. pub，一个私钥 id_dsa，需要把公钥复制到其他 slave(客户端)上去。

```
2013年 03月 24日 星期日 21:42:06 CST
[hadoop@master ~]$ ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_dsa):
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_dsa.
Your public key has been saved in /home/hadoop/.ssh/id_dsa.pub.
The key fingerprint is:
3d:d9:a6:39:db:8d:96:47:56:2a:dd:04:75:9c:6e:14 hadoop@master
[hadoop@master ~]$ cd .ssh/
[hadoop@master .ssh]$ ll
总计 16
-rw------- 1 hadoop hadoop 672 03-24 21:43 id_dsa
-rw-r--r-- 1 hadoop hadoop 603 03-24 21:43 id_dsa.pub
[hadoop@master .ssh]$ cd ~
[hadoop@master ~]$ ll
总计 0
```

图 4-87　使用 hadoop 用户执行

3）scp id_dsa. pub slave *： ~/. ssh/master. pub

其中的 slave * 代表的是每台 slave 的主机名，如图 4-88 所示。

```
[hadoop@master .ssh]$ scp id_dsa.pub slave1:`pwd`/master.pub
The authenticity of host 'slave1 (192.168.0.102)' can't be established.
RSA key fingerprint is c3:c0:1c:b0:30:e7:83:23:32:54:d8:db:48:0f:80:75.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'slave1,192.168.0.102' (RSA) to the list of known hosts.
hadoop@slave1's password:
Permission denied, please try again.
hadoop@slave1's password:
id_dsa.pub
[hadoop@master .ssh]$
```

图 4-88　scp id_dsa. pub slave *： ~/. ssh/master. pub

4）然后需要合并文件，就是把公钥文件 id_dsa. pub 写入到 authorized_keys 文件里，这样服务器才可以验证密码的有效性，如图 4-89 所示：

cat id_dsa. pub ＞＞authorized_keys

(只在 master 上操作)。

```
[hadoop@slave1 .ssh]$ cat master.pub >> authorized_keys
[hadoop@slave1 .ssh]$ ll
总计 32
-rw-rw-r-- 1 hadoop hadoop 603 03-24 21:59 authorized_keys
-rw------- 1 hadoop hadoop 668 03-24 21:55 id_dsa
-rw-r--r-- 1 hadoop hadoop 603 03-24 21:55 id_dsa.pub
-rw-r--r-- 1 hadoop hadoop 603 03-24 21:58 master.pub
```

图 4-89　合并文件

5）然后查看 authorized_keys 文件的权限，如果不是 600，修改为 600：

chmod 600 authorized_keys

（三台机器都需要操作）。

然后依次登录到其他 slave 上：

cd ~/. ssh

cat master. pub > >authorized_keys

6）然后查看 authorized_keys 文件的权限，如果不是 600，同样修改为 600，如图 4-90 所示。

```
[hadoop@slave1 .ssh]$ chmod 600 authorized_keys
[hadoop@slave1 .ssh]$ ll
总计 32
-rw------- 1 hadoop hadoop 603 03-24 21:59 authorized_keys
-rw------- 1 hadoop hadoop 668 03-24 21:55 id_dsa
-rw-r--r-- 1 hadoop hadoop 603 03-24 21:55 id_dsa.pub
-rw-r--r-- 1 hadoop hadoop 603 03-24 21:58 master.pub
[hadoop@slave1 .ssh]$
```

图 4-90　查看 authorized_keys 文件的权限

7）最后回到 master 上，如图 4-91 所示：

ssh slave *

验证是否不用输入密码便可以登录到该 slave * 上。验证成功后退出登录的服务器。

```
[hadoop@master .ssh]$ ssh slave1
Last login: Sun Mar 24 21:54:51 2013 from 192.168.0.99
```

图 4-91　回到 master

8）把 3 个包，即 hadoop-1. 0. 1. tar、zookeeper-3. 3. 5. tar 和 hbase-0. 92. 1. tar 传到 master 上的 hadoop 用户名录下/home/hadoop/用传输工具复制，之后查看是否复制成功，如图 4-92 和图 4-93 所示。

```
[hadoop@master hadoop-1.0.1]$ cd ..
[hadoop@master ~]$ ll
总计 8
drwxrwxr-x 15 hadoop hadoop 4096 03-24 22:12 hadoop-1.0.1
[hadoop@master ~]$ cd hadoop-1.0.1/
[hadoop@master hadoop-1.0.1]$ ll
总计 7880
drwxrwxr-x  2 hadoop hadoop    4096 03-24 22:11 bin
-rw-rw-r--  1 hadoop hadoop  118515 03-24 22:11 build.xml
drwxrwxr-x  4 hadoop hadoop    4096 03-24 22:11 c++
-rw-rw-r--  1 hadoop hadoop  440970 03-24 22:11 CHANGES.txt
drwxrwxr-x  2 hadoop hadoop    4096 03-24 22:11 conf
drwxrwxr-x 10 hadoop hadoop    4096 03-24 22:11 contrib
drwxrwxr-x  7 hadoop hadoop    4096 03-24 22:11 docs
-rw-rw-r--  1 hadoop hadoop    6840 03-24 22:11 hadoop-ant-1.0.1.jar
-rw-rw-r--  1 hadoop hadoop     409 03-24 22:11 hadoop-client-1.0.1.jar
-rw-rw-r--  1 hadoop hadoop 3908738 03-24 22:11 hadoop-core-1.0.1.jar
-rw-rw-r--  1 hadoop hadoop  142465 03-24 22:11 hadoop-examples-1.0.1.jar
-rw-rw-r--  1 hadoop hadoop     413 03-24 22:11 hadoop-minicluster-1.0.1.jar
-rw-rw-r--  1 hadoop hadoop 2631163 03-24 22:11 hadoop-test-1.0.1.jar
-rw-rw-r--  1 hadoop hadoop  151768 03-24 22:11 hadooptest.jar
-rw-rw-r--  1 hadoop hadoop  287807 03-24 22:11 hadoop-tools-1.0.1.jar
-rw-rw-r--  1 hadoop hadoop   10613 03-24 22:11 hbasetest.jar
drwxrwxr-x  2 hadoop hadoop    4096 03-24 22:11 ivy
-rw-rw-r--  1 hadoop hadoop   10402 03-24 22:11 ivy.xml
drwxrwxr-x  5 hadoop hadoop    4096 03-24 22:11 lib
drwxrwxr-x  2 hadoop hadoop    4096 03-24 22:11 libexec
-rw-rw-r--  1 hadoop hadoop   13366 03-24 22:11 LICENSE.txt
drwxrwxr-x  2 hadoop hadoop    4096 03-24 22:11 logs
-rw-rw-r--  1 hadoop hadoop     101 03-24 22:11 NOTICE.txt
-rw-rw-r--  1 hadoop hadoop   99898 03-24 22:11 rbase.jar
-rw-rw-r--  1 hadoop hadoop    1366 03-24 22:11 README.txt
drwxrwxr-x  2 hadoop hadoop    4096 03-24 22:11 sbin
drwxrwxr-x  3 hadoop hadoop    4096 03-24 22:12 share
drwxrwxr-x 16 hadoop hadoop    4096 03-24 22:14 src
drwxrwxr-x  9 hadoop hadoop    4096 03-24 22:12 webapps
[hadoop@master hadoop-1.0.1]$
```

图 4-92　传 3 个包

```
[hadoop@master hadoop-1.0.1]$ cd conf/
[hadoop@master conf]$ ll
总计 140
-rw-rw-r-- 1 hadoop hadoop 7457 03-24 22:11 capacity-scheduler.xml
-rw-rw-r-- 1 hadoop hadoop  535 03-24 22:11 configuration.xsl
-rw-rw-r-- 1 hadoop hadoop  530 03-24 22:11 core-site.xml
-rw-rw-r-- 1 hadoop hadoop  327 03-24 22:11 fair-scheduler.xml
-rw-rw-r-- 1 hadoop hadoop 2243 03-24 22:11 hadoop-env.sh
-rw-rw-r-- 1 hadoop hadoop 1488 03-24 22:11 hadoop-metrics2.properties
-rw-rw-r-- 1 hadoop hadoop 4644 03-24 22:11 hadoop-policy.xml
-rw-rw-r-- 1 hadoop hadoop 1019 03-24 22:11 hdfs-site.xml
-rw-rw-r-- 1 hadoop hadoop 4441 03-24 22:11 log4j.properties
-rw-rw-r-- 1 hadoop hadoop 2033 03-24 22:11 mapred-queue-acls.xml
-rw-rw-r-- 1 hadoop hadoop 1381 03-24 22:11 mapred-site.xml
-rw-rw-r-- 1 hadoop hadoop    6 03-24 22:11 masters
-rw-rw-r-- 1 hadoop hadoop   13 03-24 22:11 slaves
-rw-rw-r-- 1 hadoop hadoop 1243 03-24 22:11 ssl-client.xml.example
-rw-rw-r-- 1 hadoop hadoop 1195 03-24 22:11 ssl-server.xml.example
-rw-rw-r-- 1 hadoop hadoop  382 03-24 22:11 taskcontroller.cfg
[hadoop@master conf]$ cd ~
[hadoop@master ~]$ ll
```

图 4-93　查看

9）更改权限 chmod -R hadoop-1. 0. 1/，cd hadoop-1. 0. 1/，ll，如图 4-94 所示。

```
[hadoop@master ~]$ chmod -R 700 hadoop-1.0.1/
[hadoop@master ~]$ cd hadoop-1.0.1/
[hadoop@master hadoop-1.0.1]$ ll
总计 7880
drwx------  2 hadoop hadoop    4096 03-24 22:11 bin
-rwx------  1 hadoop hadoop  118515 03-24 22:11 build.xml
drwx------  4 hadoop hadoop    4096 03-24 22:11 c++
-rwx------  1 hadoop hadoop  440970 03-24 22:11 CHANGES.txt
drwx------  2 hadoop hadoop    4096 03-24 22:11 conf
drwx------ 10 hadoop hadoop    4096 03-24 22:11 contrib
drwx------  7 hadoop hadoop    4096 03-24 22:11 docs
-rwx------  1 hadoop hadoop    6840 03-24 22:11 hadoop-ant-1.0.1.jar
-rwx------  1 hadoop hadoop     409 03-24 22:11 hadoop-client-1.0.1.jar
-rwx------  1 hadoop hadoop 3908738 03-24 22:11 hadoop-core-1.0.1.jar
-rwx------  1 hadoop hadoop  142465 03-24 22:11 hadoop-examples-1.0.1.ja
-rwx------  1 hadoop hadoop     413 03-24 22:11 hadoop-minicluster-1.0.1
-rwx------  1 hadoop hadoop 2631163 03-24 22:11 hadoop-test-1.0.1.jar
-rwx------  1 hadoop hadoop  151768 03-24 22:11 hadooptest.jar
-rwx------  1 hadoop hadoop  287807 03-24 22:11 hadoop-tools-1.0.1.jar
-rwx------  1 hadoop hadoop   10613 03-24 22:11 hbasetest.jar
drwx------  2 hadoop hadoop    4096 03-24 22:11 ivy
-rwx------  1 hadoop hadoop   10402 03-24 22:11 ivy.xml
drwx------  5 hadoop hadoop    4096 03-24 22:11 lib
drwx------  2 hadoop hadoop    4096 03-24 22:11 libexec
-rwx------  1 hadoop hadoop   13366 03-24 22:11 LICENSE.txt
drwx------  2 hadoop hadoop    4096 03-24 22:11 logs
-rwx------  1 hadoop hadoop     101 03-24 22:11 NOTICE.txt
-rwx------  1 hadoop hadoop   99898 03-24 22:11 rbase.jar
-rwx------  1 hadoop hadoop    1366 03-24 22:11 README.txt
drwx------  2 hadoop hadoop    4096 03-24 22:11 sbin
drwx------  3 hadoop hadoop    4096 03-24 22:12 share
drwx------ 16 hadoop hadoop    4096 03-24 22:14 src
drwx------  9 hadoop hadoop    4096 03-24 22:12 webapps
```

图 4-94　更改权限

继续查看下一层，cd conf/，ll，如图 4-95 所示。

```
[hadoop@master hadoop-1.0.1]$ cd conf/
[hadoop@master conf]$ ll
总计 140
-rwx------ 1 hadoop hadoop 7457 03-24 22:11 capacity-scheduler.xml
-rwx------ 1 hadoop hadoop  535 03-24 22:11 configuration.xsl
-rwx------ 1 hadoop hadoop  530 03-24 22:11 core-site.xml
-rwx------ 1 hadoop hadoop  327 03-24 22:11 fair-scheduler.xml
-rwx------ 1 hadoop hadoop 2243 03-24 22:11 hadoop-env.sh
-rwx------ 1 hadoop hadoop 1488 03-24 22:11 hadoop-metrics2.properties
-rwx------ 1 hadoop hadoop 4644 03-24 22:11 hadoop-policy.xml
-rwx------ 1 hadoop hadoop 1019 03-24 22:11 hdfs-site.xml
-rwx------ 1 hadoop hadoop 4441 03-24 22:11 log4j.properties
-rwx------ 1 hadoop hadoop 2033 03-24 22:11 mapred-queue-acls.xml
-rwx------ 1 hadoop hadoop 1381 03-24 22:11 mapred-site.xml
-rwx------ 1 hadoop hadoop    6 03-24 22:11 masters
-rwx------ 1 hadoop hadoop   13 03-24 22:11 slaves
-rwx------ 1 hadoop hadoop 1243 03-24 22:11 ssl-client.xml.example
-rwx------ 1 hadoop hadoop 1195 03-24 22:11 ssl-server.xml.example
-rwx------ 1 hadoop hadoop  382 03-24 22:11 taskcontroller.cfg
```

图 4-95　继续查看下一层

10）压缩 hadoop 安装包到 slave1 和 slave2 上，先打包 tar -cvf hadoop-1. 0. 1. tar hadoop-1. 0. 1/，如图 4-96 所示。

```
[hadoop@master ~]$ tar -cvf hadoop-1.0.1.tar hadoop-1.0.1/
```

图 4-96　先打包 tar -cvf hadoop-1. 0. 1. tar hadoop-1. 0. 1/

查看是否压缩成功，如图 4-97 所示。

```
[hadoop@master ~]$ ll
总计 183488
drwx------ 15 hadoop hadoop      4096 03-24 22:12 hadoop-1.0.1
-rw-rw-r--  1 hadoop hadoop 187688960 03-24 22:31 hadoop-1.0.1.tar
```

图 4-97　查看是否压缩成功

复制给两个 slave，scp hadoop-1. 0. 1. tar slave *：’ pwd’ /，如图 4-98 所示。

```
-bash: spc: command not found
[hadoop@master ~]$ scp hadoop-1.0.1.tar slave1:`pwd`/
hadoop-1.0.1.tar
[hadoop@master ~]$ scp hadoop-1.0.1.tar slave2:`pwd`/
hadoop-1.0.1.tar
```

图 4-98　复制给两个 slave

11）然后在 slave1 上操作，解压 cd ～，ll，tar -xvf hadoop-1. 0. 1. tar，如图 4-99 所示。

```
[hadoop@slave1 .ssh]$ cd ~
[hadoop@slave1 ~]$ ll
总计 183480
-rw-rw-r-- 1 hadoop hadoop 187688960 03-24 22:32 hadoop-1.0.1.tar
[hadoop@slave1 ~]$ tar -xvf hadoop-1.0.1.tar
hadoop-1.0.1/
```

图 4-99　解压 cd ～，ll，tar -xvf hadoop-1. 0. 1. tar

12）解压每个包，解压之后，在每台机器上编辑 ~/. bash_profile 文件，如图 4-100 所示，末尾添加如下行：

export HADOOP_HOME = /home/hadoop/hadoop-1. 0. 1

export PATH = $ PATH：$ HADOOP_HOME/bin

exportCLASSPATH = . ：$ JAVA_HOME/lib：$ HADOOP_HOME/lib

export HADOOP_HOME_WARN_SUPPRESS = 1

```
[hadoop@master ~]$ rm hadoop-1.0.1.tar
[hadoop@master ~]$ vim ~/.bash_profile
```

图 4-100　在每台机器上编辑 ~/. bash_profile 文件

13）经过以上两个步骤，Hadoop 的安装和配置已经安成了，那么下面就来启动 Hadoop 集群。在启动 Hadoop 之前，首先要做的就是格式化 HDFS 文件目录：

hadoop namenode -format

成功情况下系统输出如图 4-101 所示。

14）启动 Hadoop 集群：

start-all. sh

系统输出如图 4-102 所示。

```
  dfsadmin             run a DFS admin client
  mradmin              run a Map-Reduce admin client
  fsck                 run a DFS filesystem checking utility
  fs                   run a generic filesystem user client
  balancer             run a cluster balancing utility
  fetchdt              fetch a delegation token from the NameNode
  jobtracker           run the MapReduce job Tracker node
  pipes                run a Pipes job
  tasktracker          run a MapReduce task Tracker node
  historyserver        run job history servers as a standalone daemon
  job                  manipulate MapReduce jobs
  queue                get information regarding JobQueues
  version              print the version
  jar <jar>            run a jar file
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create a hadoop archive
  classpath            prints the class path needed to get the
                       Hadoop jar and the required libraries
  daemonlog            get/set the log level for each daemon
 or
  CLASSNAME            run the class named CLASSNAME
Most commands print help when invoked w/o parameters.
[hadoop@master ~]$ hadoop namenode -format
13/03/24 22:38:47 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = master/192.168.0.101
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 1.0.1
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch-1.0 -r 1243785; compiled by 'hortonfo' on Tue Feb 14 08
************************************************************/
13/03/24 22:38:47 INFO util.GSet: VM type       = 32-bit
13/03/24 22:38:47 INFO util.GSet: 2% max memory = 17.77875 MB
13/03/24 22:38:47 INFO util.GSet: capacity      = 2^22 = 4194304 entries
13/03/24 22:38:47 INFO util.GSet: recommended=4194304, actual=4194304
13/03/24 22:38:47 INFO namenode.FSNamesystem: fsOwner=hadoop
13/03/24 22:38:47 INFO namenode.FSNamesystem: supergroup=supergroup
13/03/24 22:38:47 INFO namenode.FSNamesystem: isPermissionEnabled=true
13/03/24 22:38:47 INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
13/03/24 22:38:47 INFO namenode.FSNamesystem: isAccessTokenEnabled=false accessKeyUpdateInterval=0 min(s), accessTokenLifetime=0 min(s)
13/03/24 22:38:47 INFO namenode.NameNode: Caching file names occuring more than 10 times
13/03/24 22:38:47 INFO common.Storage: Image file of size 112 saved in 0 seconds.
13/03/24 22:38:47 INFO common.Storage: Storage directory /hadoop/hdfs/name has been successfully formatted.
13/03/24 22:38:47 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at master/192.168.0.101
************************************************************/
```

图 4-101　成功情况下系统输出

```
[hadoop@master ~]$ cd /hadoop/
[hadoop@master hadoop]$ ll
总计 8
drwxrwxr-x 3 hadoop hadoop 4096 03-24 22:38 hdfs
[hadoop@master hadoop]$ cd ~
[hadoop@master ~]$ ll
总计 8
drwx------ 15 hadoop hadoop 4096 03-24 22:12 hadoop-1.0.1
[hadoop@master ~]$ start-all.sh
starting namenode, logging to /home/hadoop/hadoop-1.0.1/libexec/../logs/hadoop-hadoop-namenode-master.out
slave1: starting datanode, logging to /home/hadoop/hadoop-1.0.1/libexec/../logs/hadoop-hadoop-datanode-slave1.out
slave2: starting datanode, logging to /home/hadoop/hadoop-1.0.1/libexec/../logs/hadoop-hadoop-datanode-slave2.out
The authenticity of host 'master (192.168.0.101)' can't be established.
RSA key fingerprint is 48:29:51:9d:3a:c0:9a:58:b5:c4:a8:8a:a7:f7:b3:b5.
Are you sure you want to continue connecting (yes/no)? yes
master: Warning: Permanently added 'master,192.168.0.101' (RSA) to the list of known hosts.
master: starting secondarynamenode, logging to /home/hadoop/hadoop-1.0.1/libexec/../logs/hadoop-hadoop-secondarynamenode-mas
starting jobtracker, logging to /home/hadoop/hadoop-1.0.1/libexec/../logs/hadoop-hadoop-jobtracker-master.out
slave1: starting tasktracker, logging to /home/hadoop/hadoop-1.0.1/libexec/../logs/hadoop-hadoop-tasktracker-slave1.out
slave2: starting tasktracker, logging to /home/hadoop/hadoop-1.0.1/libexec/../logs/hadoop-hadoop-tasktracker-slave2.out
k[hadoop@master ~]$ j
```

图 4-102　启动 Hadoop 集群系统输出

15）启动完成后，最好使用 jps 命令查看 Hadoop 的进程，在 master 服务节点上有如下进程如图 4-103 所示。

NameNode

JobTracker

SecondaryNameNode

在每台 slave 上有如下的进程如图 4-104 所示：

TaskTracker

DataNode

```
k[hadoop@master ~]$ jps
5507 Jps
5409 JobTracker
5141 NameNode
5322 SecondaryNameNode
[hadoop@master ~]$
```

图 4-103　master 服务节点上的进行

```
192.168.0.101   192.168.0.102   192.168.0.103
Last login: Sun Mar 24 21:59:48 2013 from master
[hadoop@slave1 ~]$ jps
4797 DataNode
4968 Jps
4889 TaskTracker
[hadoop@slave1 ~]$
```

图 4-104　每台 slave 上的进程

如上所有进程都在的话，则表示集群已经完全启动。

注：当集群启动完成后，如果还想进行其他操作，必须等待防火墙关闭，查看防火墙是否关闭的命令是：hadoop dfsadmin-safemode get。当输出是“OFF”时才可以进行操作，否则不允许操作，如图 4-105 所示。

```
[hadoop@master logs]$ hadoop dfsadmin -safemode get
Safe mode is ON
[hadoop@master logs]$ hadoop dfsadmin -safemode get
Safe mode is ON
[hadoop@master logs]$ hadoop dfsadmin -safemode get
Safe mode is ON
[hadoop@master logs]$ hadoop dfsadmin -safemode get
Safe mode is ON
[hadoop@master logs]$ hadoop dfsadmin -safemode get
Safe mode is OFF
```

图 4-105　必须是“OFF”

16）停止 Hadoop 集群：

stop-all. sh

3. Hadoop 核心程序配置

1）主要修改 cd hadoop-1. 0. 4/conf/，ll 目录下的东西，如图 4-106 所示。

```
Last login: Sun Mar 24 23:14:33 2013 from 192.168.0.99
[hadoop@master ~]$ cd hadoop-1.0.4/conf/
[hadoop@master conf]$ ll
总计 140
-rwx------ 1 hadoop hadoop 7457 10-03 13:17 capacity-scheduler.xml
-rwx------ 1 hadoop hadoop  535 10-03 13:17 configuration.xsl
-rwx------ 1 hadoop hadoop  269 03-24 23:15 core-site.xml
-rwx------ 1 hadoop hadoop  327 10-03 13:17 fair-scheduler.xml
-rwx------ 1 hadoop hadoop 2237 10-03 13:17 hadoop-env.sh
-rwx------ 1 hadoop hadoop 1488 10-03 13:17 hadoop-metrics2.properties
-rwx------ 1 hadoop hadoop 4644 10-03 13:17 hadoop-policy.xml
-rwx------ 1 hadoop hadoop  178 10-03 13:17 hdfs-site.xml
-rwx------ 1 hadoop hadoop 4441 10-03 13:17 log4j.properties
-rwx------ 1 hadoop hadoop 2033 10-03 13:17 mapred-queue-acls.xml
-rwx------ 1 hadoop hadoop  178 10-03 13:17 mapred-site.xml
-rwx------ 1 hadoop hadoop   10 10-03 13:17 masters
-rwx------ 1 hadoop hadoop   10 10-03 13:17 slaves
-rwx------ 1 hadoop hadoop 1243 10-03 13:17 ssl-client.xml.example
-rwx------ 1 hadoop hadoop 1195 10-03 13:17 ssl-server.xml.example
-rwx------ 1 hadoop hadoop  382 10-03 13:17 taskcontroller.cfg
```

图 4-106　修改

2）$HADOOP_HOME/conf/core-site. xml 是 Hadoop 的核心配置文件，对应并覆盖 core-default. xml 中的配置项。一般在这个文件中增加如下配置：cd hadoop-1. 0. 4/conf/，vim core-site. xml。

Core-site. xml 代码如下：

```
<configuration>
        <property>
                <! --用于 dfs 命令模块中指定默认的文件系统协议 -->
                <name>fs. default. name</name>
                <value>hdfs://master:9000</value>
        </property>
</configuration>
```

3）$HADOOP_HOME/conf/hdfs-site. xml 是 HDFS 的配置文件，对应并覆盖 hdfs-site. xml 中的配置项。一般在这个文件中增加如下配置：cd hadoop-1. 0. 4/conf/，vim hdfs-site. xml。

Hdfs-site. xml 代码如下：

```
<configuration>
    <property>
        <! -- DFS 中存储文件命名空间信息的目录 -->
        <name>dfs. name. dir</name>
        <value>/hadoop/data/dfs. name. dir</value>
    </property>
    <property>
        <! -- DFS 中存储文件数据的目录 -->
        <name>dfs. data. dir</name>
        <value>/hadoop/data/dfs. data. dir</value>
    </property>
    <property>
        <! --是否对 DFS 中的文件进行权限控制(测试中一般用 false)-->
        <name>dfs. permissions</name>
        <value>false</value>
    </property>
</configuration>
```

4）$HADOOP_HOME/conf/mapred-site. xml 是 Map/Reduce 的配置文件，对应并覆盖 mapred-default. xml 中的配置项。我们一般在这个文件中增加如下配置：vim mapred-site. xml。

Mapred-site. xml 代码如下：

```
<configuration>
    <property>
        <! --用来作 JobTracker 的节点的(一般与 NameNode 保持一致）-->
        <name>mapred. job. tracker</name>
        <value>master：9001</value>
    </property>
    <property>
        <! -- map/reduce 的系统目录(使用的 HDFS 的路径）-->
        <name>mapred. system. dir</name>
        <value>/hadoop/system/mapred. system. dir</value>
    </property>
    <property>
        <! -- map/reduce 的临时目录(可使用“,”隔开,设置多重路径来分摊磁盘
    IO）-->
```

```
        <name>mapred.local.dir</name>
        <value>/hadoop/data/mapred.local.dir</value>
        </property>
</configuration>
```

主从配置：在$ HADOOP_HOME/conf 目录中存在 masters 和 slaves 这两个文件，用来作 Hadoop 的主从配置。上面已经提到了 Hadoop 主要由 NameNode/DataNode 和 JobTracker/TaskTracker 构成，在主从配置里一般将 NameNode 和 JobTracker 列为主机，其他的共为从机，于是对于此处的配置应该是：vim masters，如图 4-107 所示。

Masters 代码如下：

```
master
```

修改 slaves 文件：vim slaves，如图 4-108 所示。

图 4-107　配置 masters

图 4-108　修改 slaves

Slaves 代码如下：

```
slave1
slave2
```

如果对以上介绍的配置项做了正确的配置，那么 Hadoop 集群只差启动和初体念了，当然在$ HADOOP_HOME/conf 目录下还包括其他的一些配置文件，但那些都不是必须设置的，如果有兴趣可以自己去了解。

值得注意的是 Hadoop 集群的所有机器的配置应该保持一致，一般我们在配置完 master 后，使用 scp 将 Hadoop 目录复制到其他两台 slave 上：

scp -r hadoop-1.0.1 slave1：/home/hadoop/hadoop-1.0.1

scp -r hadoop-1.0.1 slave2：/home/hadoop/hadoop-1.0.1

4.3　获取 Hadoop 安装包

4.3.1　实验目的

1）熟悉 Hadoop 的相关信息。

2）学会如何获取最新的 Hadoop 安装包。

4.3.2　实验设备

硬件：PC。

4.3.3　实验内容

1）去网上获取 Hadoop 安装包。

2）阅读 Hadoop 相关文档。

4.3.4　实验步骤

了解和获取 Hadoop 安装包：

1）搜索 Hadoop，如图 4-109 所示。

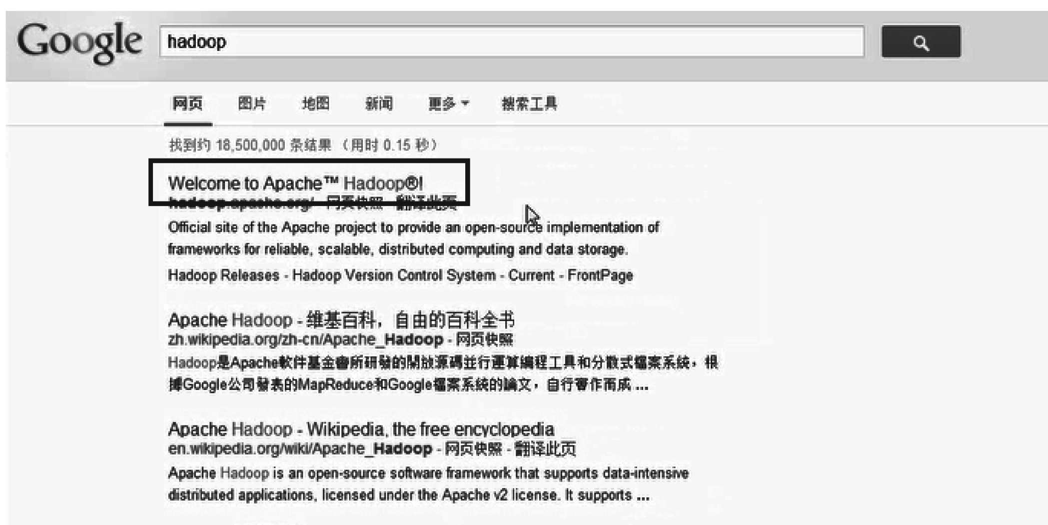


图 4-109　搜索 Hadoop

2）进入 Apache 官方网站，如图 4-110 所示。



图 4-110　进入 Apache 官方网站

3）选择稳定版本下载，如图 4-111 所示。

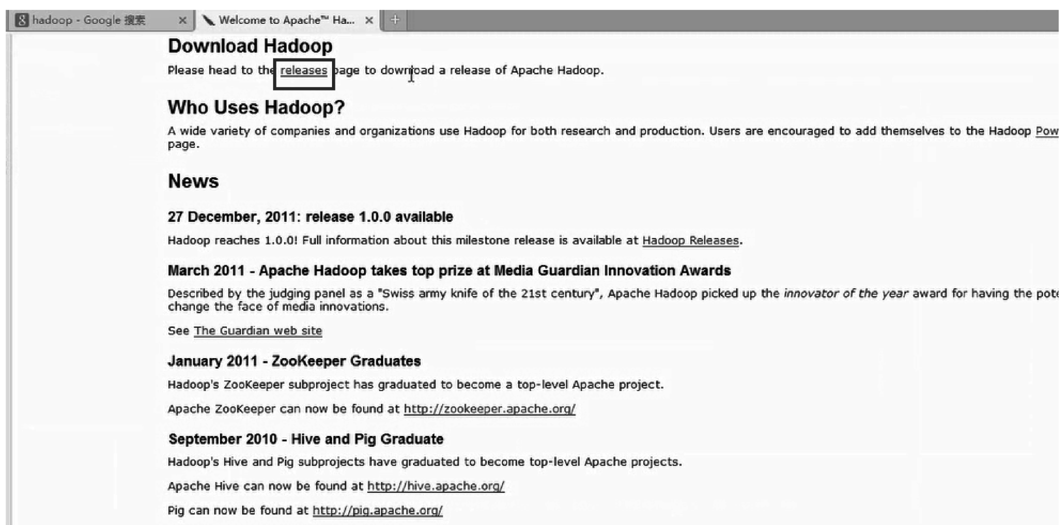


图 4-111　选择稳定版本下载

4）进入下载页，如图 4-112 所示。

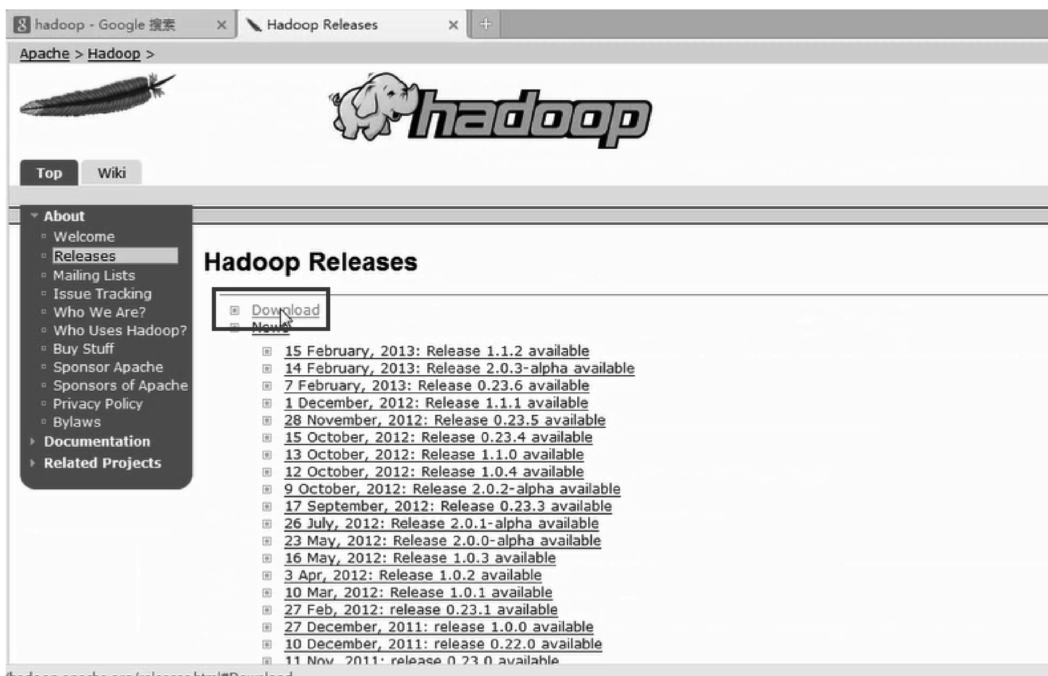


图 4-112　进入下载页

5）选择下载一个稳定版本，如图 4-113 所示。

6）选择一个合适的网址，如图 4-114 所示。

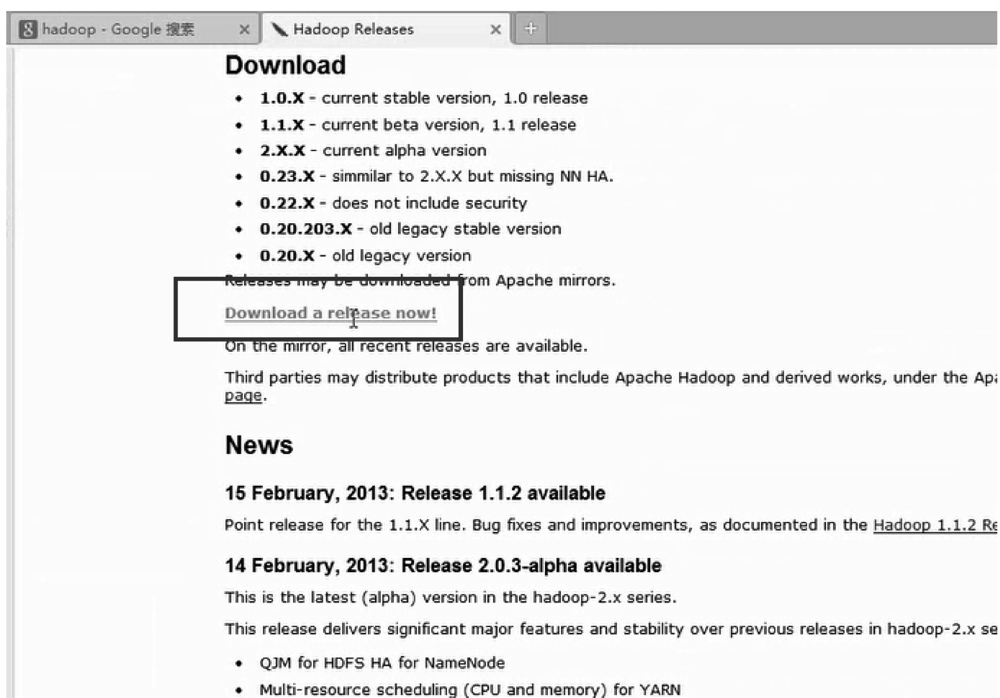


图 4-113　选择版本

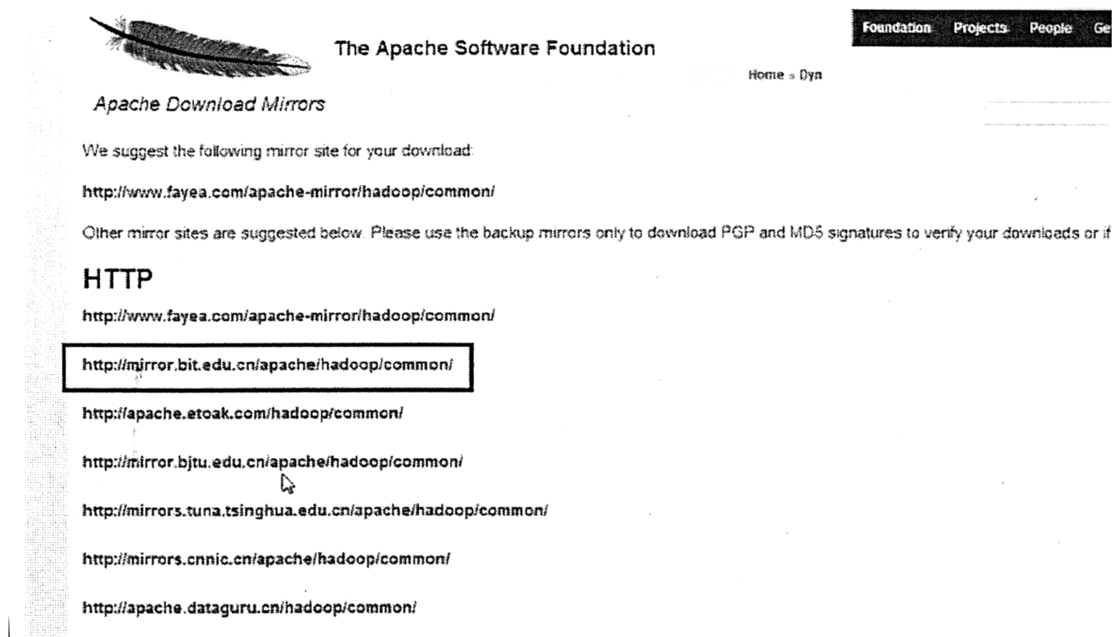


图 4-114　选择网址

7）选择稳定版，如图 4-115 所示。

8）选择 . gz 的压缩文件，进行下载，如图 4-116 所示。

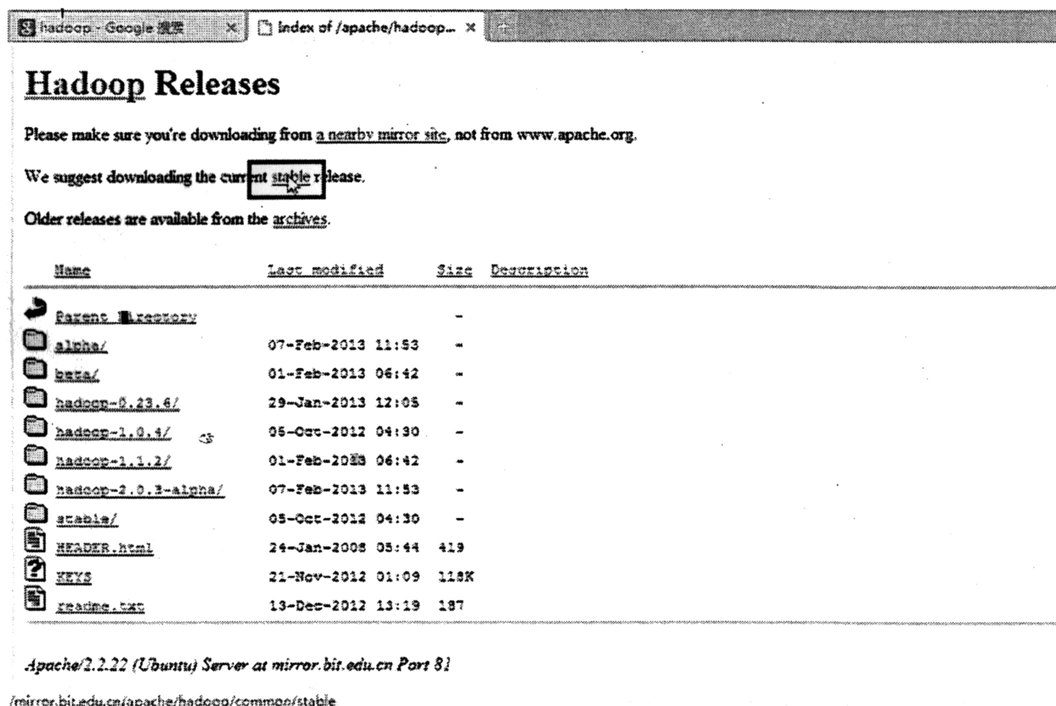


图 4-115　选择稳定版

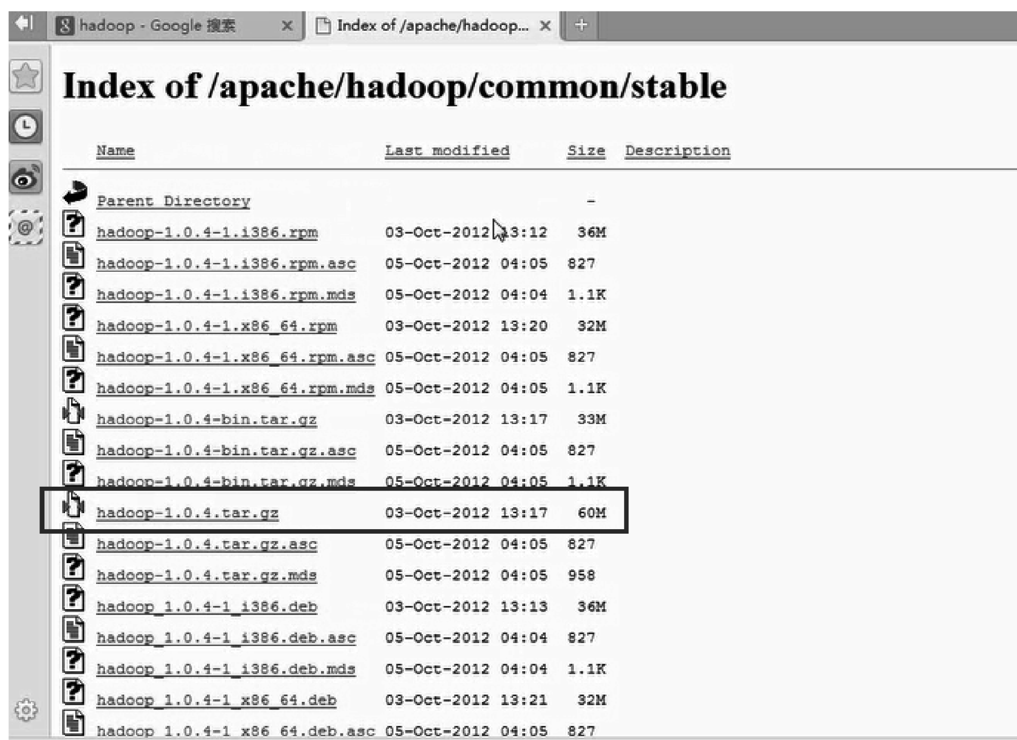


图 4-116　选择下载的压缩文件

4.4　启动和关闭 Hadoop 集群

4.4.1　实验目的

1）熟悉集群启动和关闭的步骤。
2）学会启动和关闭集群。

4.4.2　实验设备

1）硬件：云计算一体机、PC。
2）软件：SecureCRT。

4.4.3　实验内容

使用 SecureCRT 软件启动和关闭 Hadoop 集群。

4.4.4　实验步骤

1）首先使用 SecureCRT 软件连接到 master，如图 4-117 所示。
2）找到 master 的 IP 地址，如图 4-118 所示。
3）双击 IP 地址，登录成功，如图 4-119 所示。
4）首先启动 Hadoop，输入 start -all. sh，如图 4-120 所示。
5）启动 Hadoop 成功，需要等待几分钟退出安全模式，可以使用



图 4-117　使用 SecureCRT 软件

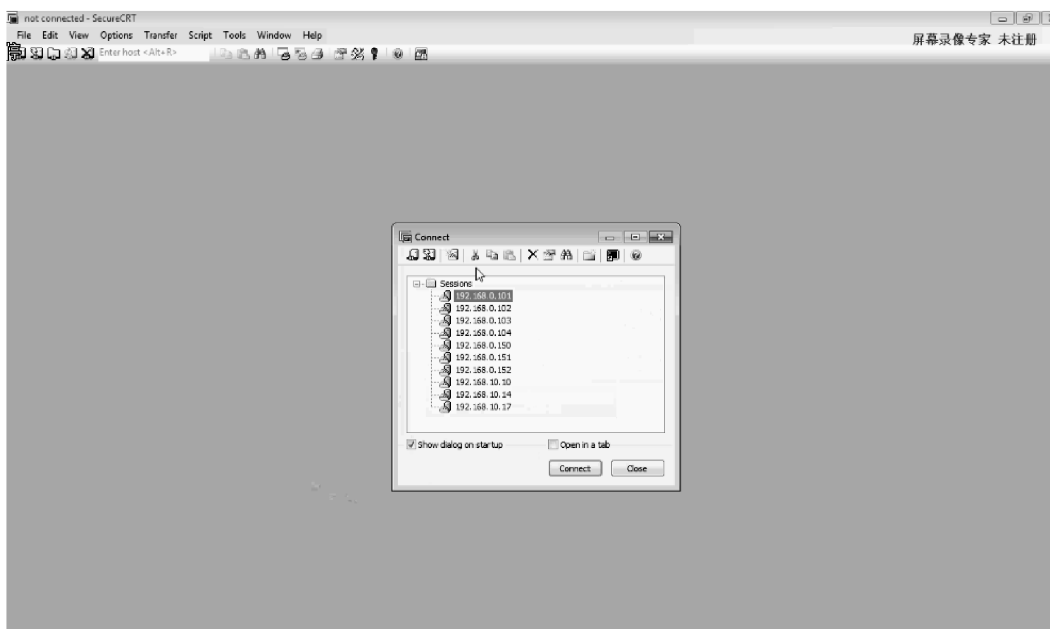


图 4-118　找到 master 的 IP 地址

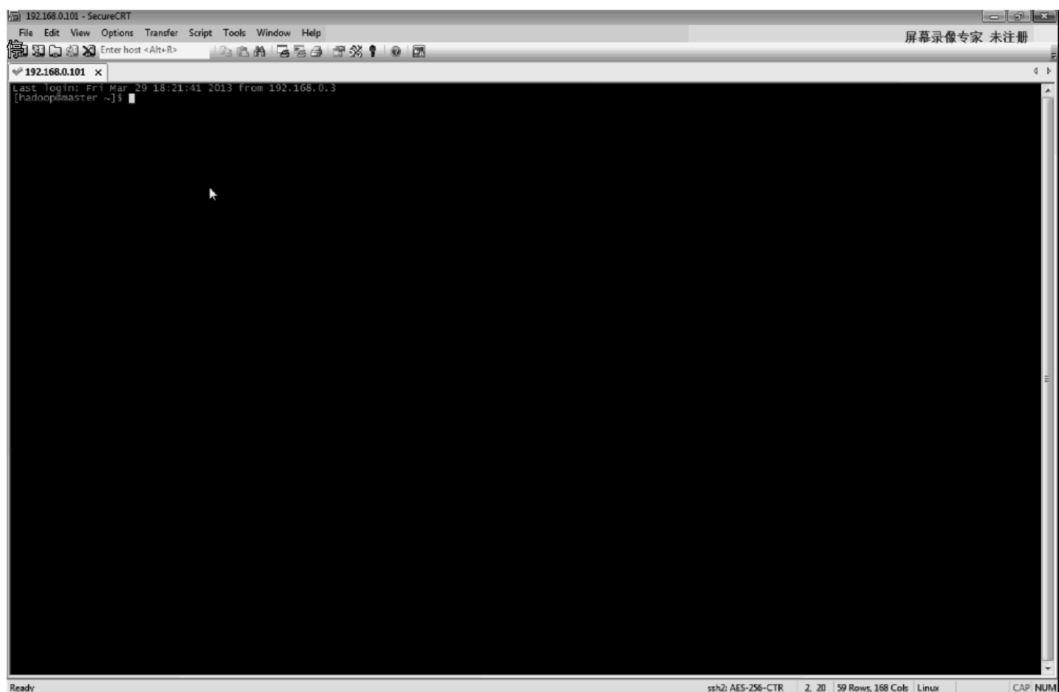


图 4-119　登录成功

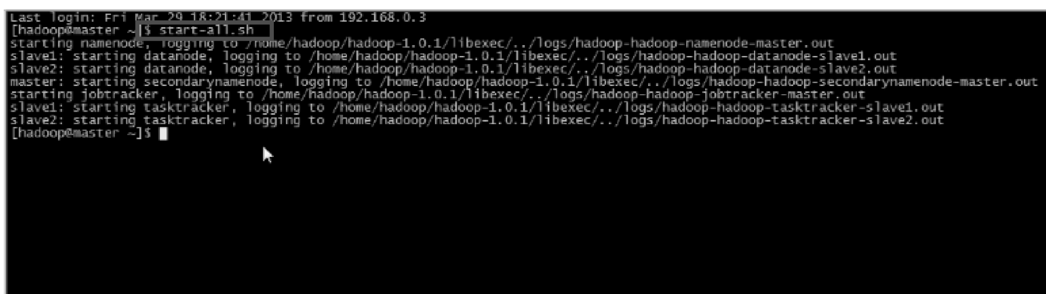


图 4-120　启动 Hadoop

hadoop dfsadmin -safemode get，命令查看时候关闭安全模式，如果是 ON，代表没有关闭，如图 4-121 所示。



图 4-121　退出安全模式

6）当出现 OFF 时，代表已经退出安全模式，如图 4-122 所示。

7）这个时候启动要启动 web 容器，首先进入 apache-tomcat-6. 0. 36 这个目录下：cd apache-tomcat-6. 0. 36/。

8）然后启动 web 容器：bin/startup. sh，如图 4-123 所示。

```
[hadoop@master ~]$ hadoop dfsadmin -safemode get
Safe mode is ON
[hadoop@master ~]$ hadoop dfsadmin -safemode get
Safe mode is ON
[hadoop@master ~]$ hadoop dfsadmin -safemode get
Safe mode is OFF
[hadoop@master ~]$
```

图 4-122　退出成功

```
[hadoop@master apache-tomcat-6.0.36]$ bin/startup.sh
Using CATALINA_BASE:   /home/hadoop/apache-tomcat-6.0.36
Using CATALINA_HOME:   /home/hadoop/apache-tomcat-6.0.36
Using CATALINA_TMPDIR: /home/hadoop/apache-tomcat-6.0.36/temp
Using JRE_HOME:        /usr/java/jdk1.6.0_30
Using CLASSPATH:       /home/hadoop/apache-tomcat-6.0.36/bin/bootstrap.jar
```

图 4-123　启动 web 容器

9）使用 jps 命令查看一下是否启动成功，如图 4-124 所示。

```
[hadoop@master apache-tomcat-6.0.36]$ jps
9970 JobTracker
9680 NameNode
10397 Bootstrap
9873 SecondaryNameNode
10420 Jps
[hadoop@master apache-tomcat-6.0.36]$
```

图 4-124　使用 jps 命令查看一下是否启动成功

10）其中 Bootstarap 是代表服务器的集成，剩下三个是 Hadoop 集群，有这几个进程代表启动成功，可以通过页面进行访问：

master. 8080/education/

输入网址，可以登录到云平台，如图 4-125 所示。



图 4-125　登录云平台

11）用户名和密码进入，如图 4-126 所示。

12）然后是关机，首先是关闭 web 容器：bin/shutdown. sh，如图 4-127 所示。

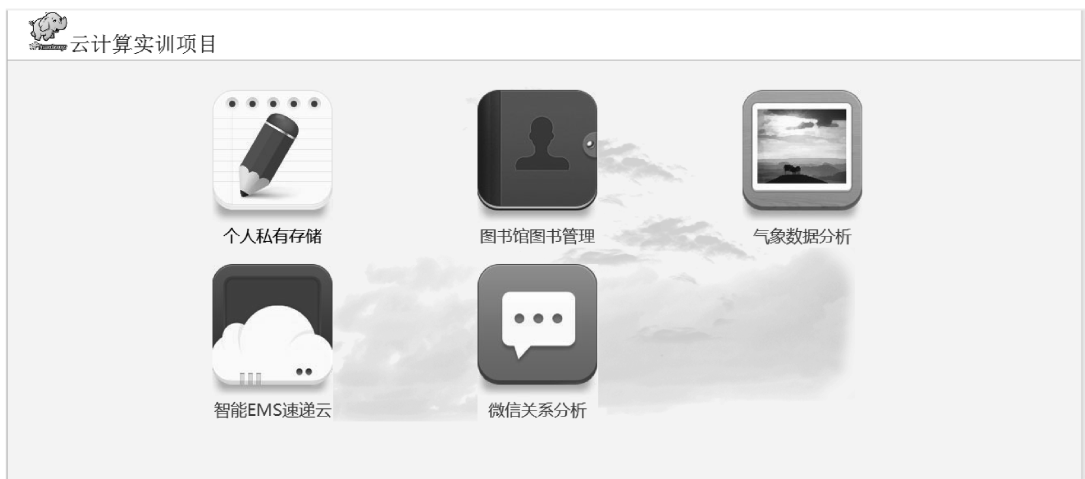


图 4-126　进入项目选择界面

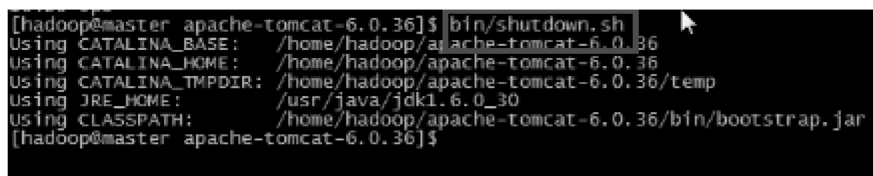


图 4-127　关闭 web 容器

13）然后是停止 Hadoop 集群：stop-all. sh，如图 4-128 所示。

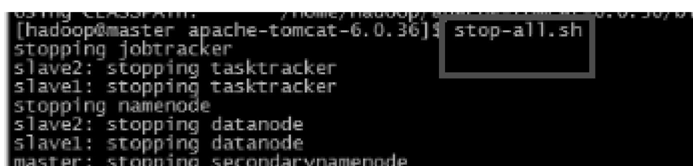


图 4-128　停止 Hadoop 集群

14）最后通过 jsp 命令看一下是否关闭成功，如图 4-129 所示。

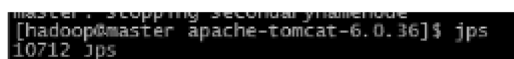


图 4-129　查看是否关闭成功

15）可以看到进程已经都没有了，表示关闭成功。这个时候就可以关闭连接工具，然后给试验箱断电。

第 5 章　熟悉 Hadoop 本地集群

5.1　熟悉 Hadoop 的一些常用命令

5.1.1　实验目的

1）熟悉 Hadoop 常用命令的使用。

2）学会在操作中使用这些命令。

5.1.2　实验设备

1）硬件：云计算一体机、PC。

2）软件：SecureCRT。

5.1.3　实验内容

使用 SecureCRT 工具在集群上操作一些常用命令。

5.1.4　实验步骤

Hadoop 常用命令：

1）预览命令 ls，hadoop dfs-ls/，如图 5-1 和图 5-2 所示。

```
[hadoop@master ~]$ hadoop dfs -ls /
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2013-03-24 22:45 /hadoop
[hadoop@master ~]$
```

图 5-1　预览命令(一)

```
[hadoop@master ~]$ hadoop dfs -ls /
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2013-03-24 22:45 /hadoop
[hadoop@master ~]$ hadoop dfs -ls /hadoop
Found 1 items
drwxr-xr-x   - hadoop supergroup          0 2013-03-24 23:29 /hadoop/mapred
[hadoop@master ~]$ hadoop dfs -ls /hadoop/mapred
Found 1 items
drwx------   - hadoop supergroup          0 2013-03-24 23:29 /hadoop/mapred/system
[hadoop@master ~]$ hadoop dfs -ls /hadoop/mapred/system
Found 1 items
-rw-------   2 hadoop supergroup          4 2013-03-24 23:29 /hadoop/mapred/system/jobtracker.info
[hadoop@master ~]$
```

图 5-2　预览命令(二)

2）传输命令 put，hadoop dfs -put，如图 5-3 所示。

```
[hadoop@master ~]$ hadoop dfs -put
Usage: java FsShell [-put <localsrc> ... <dst>]
[hadoop@master ~]$
```

图 5-3　传输命令

先放一个文件到集群上，如图 5-4 所示。

3）查看时复制到机器上 ll，如图 5-5 所示。

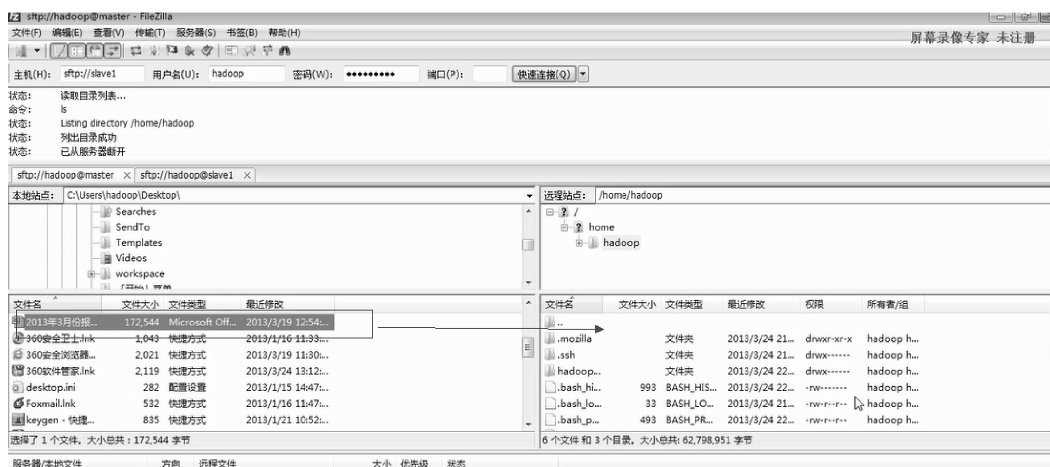


图 5-4　传输文件到集群

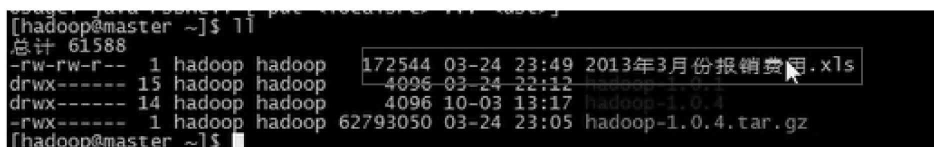


图 5-5　复制到机器上

4）复制这个文件到根目录 test 目录下，hadoop dfs-put *. */test/ *. *，如图 5-6 所示。



图 5-6　复制到根目录

5）查看 hadoop dfs-ls /test，如图 5-7 所示。



图 5-7　查看

6）页面浏览 http：//master：50070，如图 5-8 所示。后台查看复制的文件如图 5-9 所示。

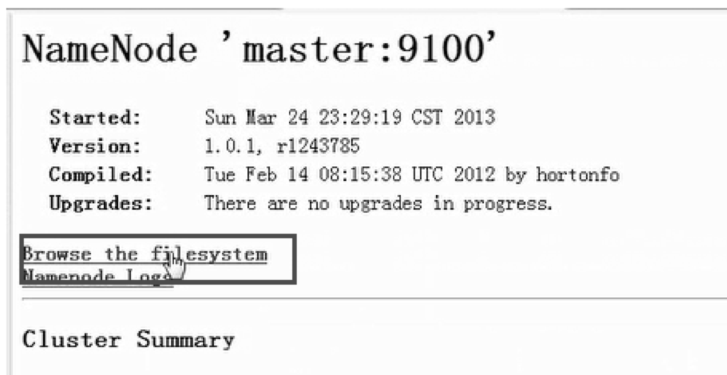


图 5-8　浏览后台文件页面

hadoop - Google 搜索　HDFS:/　22.07 GB DFS Remaining...

Contents of directory /

Goto : /　go

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
hadoop	dir				2013-03-24 22:45	rwxr-xr-x	hadoop	supergroup
test	dir				2013-03-24 23:52	rwxr-xr-x	hadoop	supergroup

Go back to DFS home

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
2013年3月份报销费用.xls	file	168.5 KB	2	64 MB	2013-03-24 23:52	rw-r--r--	hadoop	supergroup

图 5-9　后台查看复制的文件

7）删除文件命令 rm，hadoop dfs-rm /test/ *. *，如图 5-10 所示。

```
[hadoop@master ~]$ hadoop dfs -rm /test/2013年3月份报销费用.xls
Deleted hdfs://master:9100/test/2013年3月份报销费用.xls
[hadoop@master ~]$ hadoop dfs -ls /test
```

图 5-10　删除文件命令

8）删除文件夹命令 rmr，hadoop dfs-rmr /test，如图 5-11 所示。

```
[hadoop@master ~]$ hadoop dfs -rmr /test
Deleted hdfs://master:9100/test
[hadoop@master ~]$
```

图 5-11　删除文件夹命令

页面查看删除文件如图 5-12 所示。

9）上传文件，put 命令，hadoop dfs-put Text. txt /embest/text1. txt，如图 5-13 所示。

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
hadoop	dir				2013-03-24 22:45	rwxr-xr-x	hadoop	supergroup

Go back to DFS home

Local logs

图 5-12　页面查看删除文件

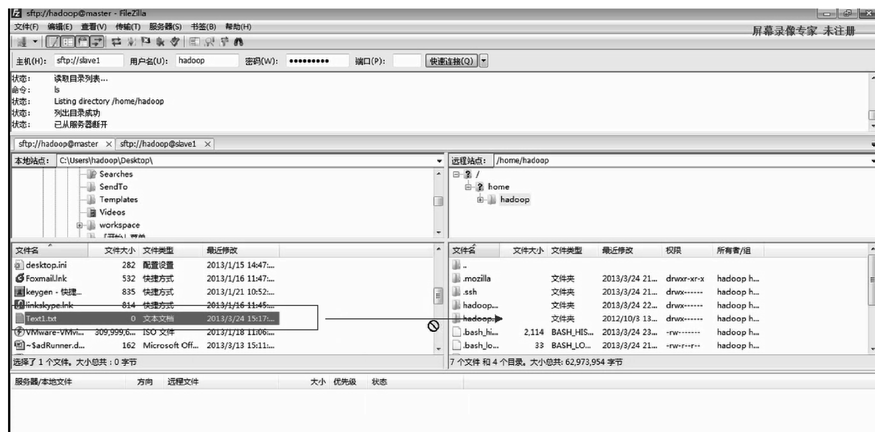


图 5-13　上传测试文件到集群

10）看一看是否传输完成 ll，如图 5-14 所示。

```
[hadoop@master ~]$ ll
总计 61592
-rw-rw-r--  1 hadoop hadoop   172544 03-24 23:49 2013年3月份报销费用.xls
drwx------ 15 hadoop hadoop     4096 03-24 22:12 hadoop-1.0.1
drwx------ 14 hadoop hadoop     4096 10-03 13:17 hadoop-1.0.4
-rwx------  1 hadoop hadoop 62793050 03-24 23:05 hadoop-1.0.4.tar.gz
-rw-rw-r--  1 hadoop hadoop        0 03-24 23:55 Text1.txt
[hadoop@master ~]$ hadoop dfs -put Text1.txt /embest/text1.txt
[hadoop@master ~]$ 
```

图 5-14　看一看是否传输完成

11）打开命令 cat，hadoop dfs-cat /embest/text1. txt，如图 5-15 所示。

```
[hadoop@master ~]$ hadoop dfs -cat /embest/text1.txt
```

图 5-15　打开命令

命令行浏览被打开的文件如图 5-16 所示，页面浏览被打开的文件如图 5-17 所示。



图 5-16　命令行浏览被打开的文件



图 5-17　页面浏览被打开的文件

12）从另一台机器上下载 get 命令，hadoop dfs-get /embest/text2. txt ~/mytxt. txt，ll，如图 5-18 所示。

```
5905 DataNode
[hadoop@slave1 ~]$ hadoop dfs -get /embest/text2.txt ~/mytxt.txt
[hadoop@slave1 ~]$ ll
总计 28
drwx------ 15 hadoop hadoop 4096 03-24 22:12 hadoop-1.0.1
drwx------ 14 hadoop hadoop 4096 03-24 23:25 hadoop-1.0.4
-rw-rw-r--  1 hadoop hadoop 4439 03-25 00:03 mytxt.txt
[hadoop@slave1 ~]$ c
```

图 5-18 get 命令

13）创建文件夹 mkdir，hadoop dfs-mkdir /books，hadoop dfs-ls/，如图 5-19 所示。

```
[hadoop@master ~]$ hadoop dfs -mkdir /books
[hadoop@master ~]$ hadoop dfs -ls /
Found 3 items
drwxr-xr-x   - hadoop supergroup          0 2013-03-25 00:10 /books
drwxr-xr-x   - hadoop supergroup          0 2013-03-25 00:08 /embest
drwxr-xr-x   - hadoop supergroup          0 2013-03-24 22:45 /hadoop
[hadoop@master ~]$ hadoop dfs -dus /embest
```

图 5-19 创建文件夹

14）看文件夹有多大空间 dus 命令，hadoop dfs-dus/embest，如图 5-20 所示。

```
[hadoop@master ~]$ hadoop dfs -dus /embest
hdfs://master:9100/embest       11876
[hadoop@master ~]$
```

图 5-20 dus 命令

5.2 使用 distcp 进行并行复制

前面的 HFDS 访问模型都集中于单线程的访问。例如通过指定文件通配，可以对一部分文件进行处理，但是为了高效，对这些文件的并行处理需要新写一个程序。Hadoop 有一个叫 distcp(分布式复制)的有用程序，能从 Hadoop 的文件系统并行复制大量数据。

Distcp 一般用于在两个 HDFS 集群中传输数据。如果集群在 Hadoop 的同一版本上运行，就适合使用 hdfs 方案：hadoop distcp hdfs：//namenodel/foo hdfs：//namenode2. bar。

这将从第一个集群中复制/foo 目录(和它的内容)到第二个集群中的/bar 目录下，所以第二个集群会有/bar/foo 目录结构。如果/bar 不存在，则新建一个。我们可以指定多个源路径，并且所有的都会被复制到目标路径。源路径必须是绝对路径。

默认情况下，distcp 会跳过目标路径已经有的文件，但可以通过提供的-overwrite 选项进行覆盖。也可以用-update 选项来选择只更新那些修改过的文件。

注意：使用-overwrite 和-update 中任意一个(或两个)选项会改变源路径和目标路径的含义。这可以用一个例子说明清楚。如果改变先前例子中第一个集群的子树/foo 下的一个文件，就能通过运行对第二个集群的改变进行同步：hadoop distcp-update hdfs：//namenode1/foo hdfs：//namenode2/bar/foo。

目标路径需要末尾这个额外的子目录/foo，因为源目录下的内容已被复制到目标目录下(如果熟悉 rsync,可以想象-overwrite 或-update 项对源路径而言,如同添加一个隐含的斜杠)。

如果对 distcp 操作不是很确定，最好先对一个小的测试目录树进行尝试。

有很多选项可以控制分布式复制行为，包括预留文件属性，忽略故障和限制复制的文件或总数据点的数量。运行时不带任何选项，可以看到使用说明。

Distcp 是作为一个 MapReduce 作业执行的，复制工作由集群中并行运行的 map 来完成。这里并没有 reducer。每个文件都由一个单一的 map 进行复制，并且 distcp 通过将文件分成大致相等的文件来给每个 map 数量大致相同的数据。

map 的数量是这样确定的，通过让每一个 map 复制数量合理的数据以最小化任务建立所涉及的开销是一个很好的想法，所以每个 map 的副本最少为 256MB(除非输入的总大小较少,否则一个 map 就足以操控全局)。例如，1GB 的文件会被分成 4 个 map 任务。如果数据很大，为限制带宽和集群的使用而限制映射的数量就会变得很有必要。map 默认的最大数量是每个集群节点(tasktracker)有 20 个。例如，复制 1000GB 的文件到一个 100 个节点的集群，会分配 2000 个 map(每个节点 20 个 map)，所以平均每个会复制 512MB。通过对 distcp 指定 -m 参数，会减少映射的分配数量。例如，-m 1000 会分配 1000 个 map，平均每个复制 1GB。

如果想在两个运行着不同版本 HDFS 的集群上利用 distcp，使用 hdfs 协议会失败，因为 RPC 系统是不兼容的。想要弥补这种情况，可以使用基于 HTTP 的 HFTP 文件系统从源中进行读取。这个作业必须运行在目标集群上，使得 HDFS RPC 版本是兼容的。使用 HFTP 重复前面的例子：hadoop distcp hftp://namenode1:50070/foo hfds://namenode2/bar。

注意，需要在 URI 源中指定名称节点的 Web 端口。这是由 dfs. http. address 的属性决定的，默认值为 50070。

5.3　Web 浏览 Hadoop 集群

5.3.1　实验目的

1）熟练掌握页面方式浏览 Hadoop 集群。

2）学会查看集群中的各个文件。

5.3.2　实验设备

硬件：云计算一体机、PC。

5.3.3　实验内容

使用浏览器访问 Hadoop 集群。

5.3.4　实验步骤

1）输入网址 http://master:50070，能看到摘要，页面访问后台文件界面，如图 5-21 所示。

2）MapReduce 后台网址 http://master:50030，也可以看到摘要，页面访问后台任务界面，如图 5-22 所示。

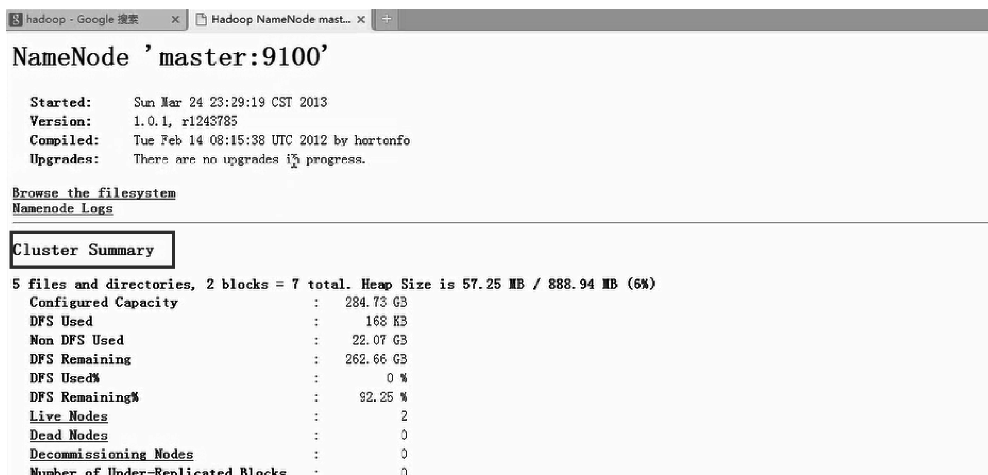
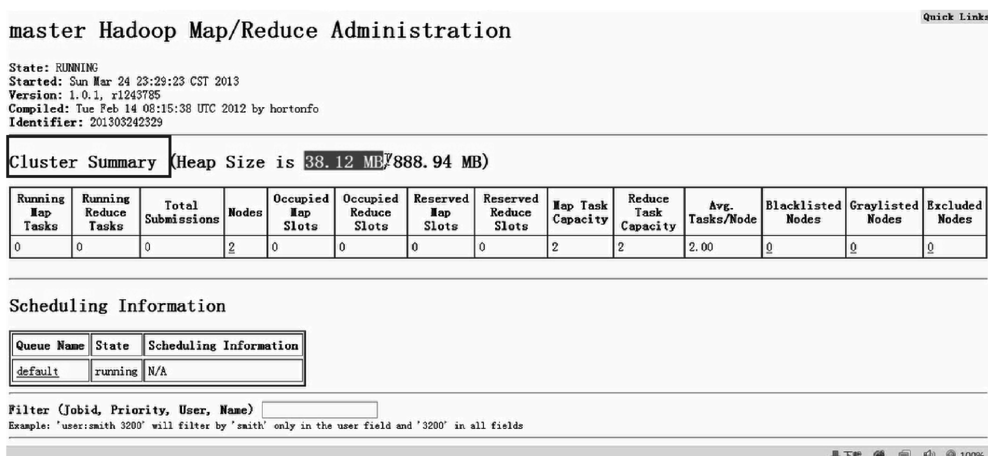


图 5-21　页面访问后台文件界面



Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	0	2	0	0	0	0	2	2	2.00	0	0	0

Queue Name	State	Scheduling Information
default	running	N/A

图 5-22　页面访问后台任务界面

5.4　使用 Hadoop 命令归档文件

1. Hadoop 归档文件

每个文件以块方式存储，块的元数据存储在名称节点的内存里，此时存储一些小的文件，HDFS 会比较低效。因此，大量的小文件会耗尽名称节点的大部分内存(注意,相较于存储文件原始内容所需要的磁盘空间,小文件所需要的空间不会更多。例如,一个 1MB 的文件以大小为 128MB 的块存储,使用的是 1MB 的磁盘空间,而不是 128MB)。

Hadoop Archives 或 HAR 文件，是一个更高效地将文件放入 HDFS 块中的文件存档设备，在减少名称节点内存使用的同时，仍然允许对文件进行透明地访问。具体说来，Hadoop Archives 可以被用作 MapReduce 的输入。

2. 使用 Hadoop Archives

Hadoop Archives 通过使用 archive 工具根据一个文件集合创建而来。这些工具运行一个 MapReduce 作业来并行处理输入文件，因此我们需要一个 MapReduce 集群去运行使用它。HDFS 中有一些我们希望归档的文件：hadoop fs-lsr/my/files，即

```
%   hadoop fs -lsr /my/files
-rw-r--r--     1 tom supergroup     1   2009-04-09   19:13
/my/files/a
drwxr-xr-x   -tom supergroup     0   2009-04-09   19:13
/my/files/dir
-rw-r--r--     1 tom supergroup     1   2009-04-09   19:13
/my/files/dir/b
```

现在我们可以运行 archive 指令：hadoop archive-archiveName files. har /my/files /my。

第一个选项是归档文件名称，这里是 file. har。HAR 文件总是有一个 . har 扩展名，这是必需的，具体理由将在后面描述。接下来把文件放入归档文件。这里只归档一个源树，即 HDFS 下/my/files 中的文件，但事实上，该工具接受多个源树。最后一个参数是 HAR 文件的输出目录。下面看看这个归档文件是怎么创建的：hadoop fs-ls /my，即

```
%   hadoop fs -ls /my
Found 2   items
Hadoop fs-ls /my/files. har，即
drwxr-xr-x   -tom supergroup     0   2009-04-09   19:13 /my/files
drwxr-xr-x   -tom supergroup     0   2009-04-09   19:13
/my/files. har
%   hadoop fs -ls /my/files. har
Found 3 items
-rw-r--r--     10 tom supergroup     165   2009-04-09   19:13
/my/files. har/_ index
-rw-r--r--     10 tom supergroup     23   2009-04-09   19:13
/my/files. har/_ masterindex
-rw-r--r--     1 tom supergroup     2   2009-04-09   19:13
/my/files. har/part-0
```

这个目录列表展示了一个 HAR 文件的组成部分：两个索引文件和部分文件的集合（本例中只有一个）。这些部分文件包括已经链接在一起的大量原始文件的内容，并且索引使我们可以查找那些包含归档文件的部分文件，包括它的起始点和长度。但所有这些细节对于使用 har URI 方案与 HAR 文件交互的应用都是隐藏的，HAR 文件系统是建立在基础文件系统上的（本例中是 HDFS）。以下命令以递归方式列出了归档文件中的文件：hadoop fs-lsr har：///my/files. har，即

```
%   hadoop fs -lsr har:///my/files. har
drw-r--r--     -tom supergroup     0   2009-04-09   19:13
/my/files. har/my
drw-r--r--     -tom supergroup     0   2009-04-09   19:13
```

/my/files. har/my/files
rw-r--r--　　10 tom supergroup　　1　2009-04-09　19:13
/my/files. har/my/files/a
drw-r--r--　　- tom supergroup　　0　2009-04-09　19:13
/my/files. har/my/files/dir
-rw-r--r--　　10 tom supergroup　　1　2009-04-09　19:13
/my/files. har/my/files/dir/b

如果 HAR 文件所在的文件系统是默认的文件系统，这就非常直观易懂。但如果想使用在其他文件系统中的 HAR 文件，就需要使用一个不同于正常情况的 URI 路径格式。以下两个指令作用相同，例如：

% hadoop fs-lsr har：///my/files. har/my/files/dir

% hadoop fs-lsr har：//hdfs-localhost：8020/my/files. har/my/files/dir

注意第二个格式，仍以 har 方案表示一个 HAR 文件系统，但是是由 hdfs 指定基础的文件系统方案，后面加上一个横杠和 HDFS host(localhost)和端口(8020)。现在算是明白为什么 HAR 文件必须要有 . har 扩展名了。通过查看权限和路径及 . har 扩展名的组成 HAR 文件系统将 har URI 转换成为一个基础文件系统的 URI。在本例中是 hdfs：/localhost：8020/user/tom/files. har。路径的剩余部分是文件在归档文件中的路径：/user/tom/files/dir。

要想删除一个 HAR 文件，需要使用删除的递归格式，因为对于基础文件系统来说，HAR 文件是一个目录：hadoop fs-rmr /my/files. har。

3. 不足

对于 HAR 文件，还需要了解它的一些不足。创建一个归档文件会创建原始文件的一个副本，因此需要与要归档(尽管创建了归档文件后可以删除原始文件)的文件同样大小的磁盘空间。虽然归档的文件能被压缩(HAR 文件在这方面像 tar 文件)，但是目前还不支持档案压缩。

一旦创建，archives 便不可改变。要增加或移除文件必须新创建归档文件。事实上，这对那些写后便不能改的文件来说不是问题，因为它们可以定期成批归档，比如每日或每周。如前所述，HAR 文件可以用作 MapReduce 的输入。然而，没有归档 InputFormat 可以打包多个文件到一个单一的 MapReduce，所以即使在 HAR 文件中处理许多小文件，也仍然低效。

第 6 章　Hadoop 管理应用

6.1　系统检查和报告

Hadoop 提供的文件系统检查工具叫做 fsck。如参数为文件路径时，它会递归地检查该路径下所有文件的健康状态。如果参数为/，它就会检查整个文件系统。图 6-1 所示为一个输出的示例：bin/hadoop fsck /。

```
[hadoop@master ~]$ hadoop fsck /
FSCK started by hadoop from /192.168.0.101 for path / at Mon Jun 10 13:04:46 CST 20
13
...............................
/hadoop/tmp/mapred/staging/hadoop/.staging/job_201303250301_0006/job.jar:  Under re
plicated blk_2729238035191764277_1065. Target Replicas is 10 but found 2 replica(s)
.
.
/hadoop/tmp/mapred/staging/hadoop/.staging/job_201303250301_0013/job.jar:  Under re
plicated blk_7056796516635261677_1127. Target Replicas is 10 but found 2 replica(s)
.
...................................................................
..........................................................................Status: H
EALTHY
 Total size:    421654591 B
 Total dirs:    114
 Total files:   174
 Total blocks (validated):      170 (avg. block size 2480321 B)
 Minimally replicated blocks:   170 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       2 (1.1764706 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    2
 Average block replication:     1.9764706
 Corrupt blocks:                0
 Missing replicas:              16 (4.7619047 %)
 Number of data-nodes:          2
 Number of racks:               1
FSCK ended at Mon Jun 10 13:04:46 CST 2013 in 57 milliseconds


The filesystem under path '/' is HEALTHY
[hadoop@master ~]$
```

图 6-1　一个输出的示例

大多数信息是不言而喻的。默认情况下，fsck 会忽略正在被客户端写入而打开的文件。如果想看到这些文件的列表，可以在 fsck 中使用-openforwrite 参数。

在 fsck 检查文件系统时，它每发现一个健康的文件就会打印出一个圆点(未在前述的输出中显示)。遇到不健康的文件时，它会打出相应信息，包括过度复制的块、复制不足的块、未复制的块、损坏的块和失踪的副本。因为 HDFS 是自我修复的，所以过度复制的块、复制不足的块、未复制的块都不足为虑。但是，损坏的块和失踪的副本意味着数据已永久丢失。默认情况下，fsck 对损坏的文件什么也不做，但可以运行 fsck 的-delete 选项将其删除。但更好的方式是用-move 选项运行 fsck，它会把已损坏的文件移动到/lost + found 目录中备用。

还可以在 fsck 中加上-files、-blocks、-locations 和-racks 选项，以打印出更多的信息。每个后续的选项需要前面选项的存在：-blocks 需要-files；-locations 既要有-files，也要有-blocks，以此类推。-files 选项让 fsck 每检查一个文件便打印一行信息，包括文件路径、文件字节数与块个数，以及文件的状态。-blocks 选项进一步让 fsck 为文件中的每个块打印出一行信息，该行包括块名、长度及其副本数。-locations 选项会让每一行包含块的副本位置。-racks 选项

则会在位置信息中增加机架名。例如，一个小到仅有一个数据块的文件会给出如下的报告：bin/hadoop fsck /user/hadoop/test-files-blocks-locations-racks，如图 6-2 所示。

```
[hadoop@master /]$ hadoop fsck / -files -blocks -locations -racks
FSCK started by hadoop from /192.168.0.101 for path / at Mon Jun 10 13:10:26 CST 2013
..............................
/hadoop/tmp/mapred/staging/hadoop/.staging/job_201303250301_0006/job.jar:  Under replicated blk_2729
.
/hadoop/tmp/mapred/staging/hadoop/.staging/job_201303250301_0013/job.jar:  Under replicated blk_7056
....................................................................
..........................................................................Status: HEALTHY
 Total size:    421654591 B
 Total dirs:    114
 Total files:   174
 Total blocks (validated):      170 (avg. block size 2480321 B)
 Minimally replicated blocks:   170 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       2 (1.1764706 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    2
 Average block replication:     1.9764706
 Corrupt blocks:                0
 Missing replicas:              16 (4.7619047 %)
 Number of data-nodes:          2
 Number of racks:               1
FSCK ended at Mon Jun 10 13:10:26 CST 2013 in 50 milliseconds


The filesystem under path '/' is HEALTHY
[hadoop@master /]$
```

图 6-2　报告

虽然 fsck 会给出 HDFS 中每个文件的报告，但获知 DataNode 的情况却需要用 dfsadmin 命令，可以使用 dfsadmin 命令中的-report 选项：bin/hadoop dfsadmin -report，如图 6-3 所示。

```
Name: 192.168.0.102:50010
Decommission Status : Normal
Configured Capacity: 152863682560 (142.37 GB)
DFS Used: 290648064 (277.18 MB)
Non DFS Used: 11959840768 (11.14 GB)
DFS Remaining: 140613193728(130.96 GB)
DFS Used%: 0.19%
DFS Remaining%: 91.99%
Last contact: Mon Jun 10 13:11:45 CST 2013


[hadoop@master /]$ hadoop dfsadmin -report
Configured Capacity: 305727365120 (284.73 GB)
Present Capacity: 282010820608 (262.64 GB)
DFS Remaining: 281562374144 (262.23 GB)
DFS Used: 448446464 (427.67 MB)
DFS Used%: 0.16%
Under replicated blocks: 2
Blocks with corrupt replicas: 0
Missing blocks: 0

-------------------------------------------------
Datanodes available: 2 (2 total, 0 dead)

Name: 192.168.0.103:50010
Decommission Status : Normal
Configured Capacity: 152863682560 (142.37 GB)
DFS Used: 157798400 (150.49 MB)
Non DFS Used: 11756601344 (10.95 GB)
DFS Remaining: 140949282816(131.27 GB)
DFS Used%: 0.1%
DFS Remaining%: 92.21%
Last contact: Mon Jun 10 13:12:15 CST 2013


Name: 192.168.0.102:50010
Decommission Status : Normal
Configured Capacity: 152863682560 (142.37 GB)
DFS Used: 290648064 (277.18 MB)
Non DFS Used: 11959943168 (11.14 GB)
DFS Remaining: 140613091328(130.96 GB)
DFS Used%: 0.19%
DFS Remaining%: 91.99%
Last contact: Mon Jun 10 13:12:15 CST 2013


[hadoop@master /]$
```

图 6-3　使用 dfsadmin 命令中的-report 选项

若要看 NameNode 的当前活动状态，可以在 dfsadmin 中使用-metasave 选项：bin/hadoop dfsadmin-metasave filename。

这会将一部分 NameNode 的元数据保存到日志目录下 filename 文件中。在这些元数据中，会发现待复制的块以及待删除块的列表。每个块也会有一个复制列表，指向它被复制到的 DataNode。最后，这个文件还会给每个 DataNode 统计摘要。

6.2　了解 HDFS 的平衡命令

1. 删除 DataNode

有时会从 HDFS 集群中删除 DataNode，例如离线升级或者维护一台机器。从 Hadoop 中删除节点是非常简单的。虽然不推荐这样做，但的确可以“杀死”节点或将之从集群中断开。HDFS 的设计非常有弹性，让一两个 DataNode 离线不会影响操作的正常运行。NameNode 会检测到节点的死亡，并开始复制那些低于约定副本数的数据块。为了让操作更为顺畅和安全，特别当删减大批 DataNode 时，应该使用 Hadoop 的退役(decommissioning)功能，该功能确保所有块在剩余的活动节点上仍达到所需的副本数。要使用此功能，必须在 NameNode 的本地文件系统上生成一个排除文件(最初是空的)，并让参数 dfs. hosts. exclide 在 NameNode 的启动过程中指向该文件。当想删除 DataNode 时，把它们列在排除文件中，每行列出一个节点。还必须用完整的主机名，IP 或 IP：port 的格式来指定节点。执行 Bin/hadoop dfsadmin-refreeNodes 来强制 Name Node 重新读取排除文件，并开始退役过程。当此过程结束后，NameNode 的日志文件会出现像“Decommission complete for node 172. 16. 1. 55：50010”这样的消息，此时就可以将节点从集群中移出了。

如果在启动 HDFS 时没有让 dfs. hosts. exclude 指向排除文件，退役 DataNode 的正确方法是关闭 NameNode，设置 dfs. hosts. exclude 指向一个空的排除文件，重新启动 NameNode。在 NameNode 成功重启后，就可以按照上面的步骤操作。请注意，如果在重启 NameNode 之前在排除文件中列出了需要删减的 DataNode，那么 NameNode 就会混淆，而在日志中出现“ProcessReport form unregistered node：node055：50010”这样的消息。NameNode 会认为它接触到的是系统之外的 DataNode，而不是即将删除的节点。

如果退役的机器在后来的某个时刻还会重新加入集群，应该在外部文件中删除它们，并重新执行 bin/hadoop dfsadmin-refreshNodes 来更新 NameNode。当机器已经准备好重新加入集群时，可以按照后面介绍的步骤来添加它们。

2. 增加 DataNode

除了让离线维护的机器重新上线，可能还会在 Hadoop 集群中增加 DataNode，以便有更多的作业来处理更多的数据。采用与集群中所有 DataNode 都一样的方式在新节点上安装 Hadoop 并设置配置文件。手动启动 DataNode 的守护进程(bin/hadoop datanode)。它会自动联系 NameNode 并加入集群。还应把新节点添加到主服务器的 conf/slaves 文件中。脚本命令会识别到新节点。

当添加一个新的 DataNode 时，它最初会是空的，然而早先的 DataNode 已经存了一些内容，这时文件系统被认为是不平衡的。新的文件将有可能进入新节点，但其副本仍会进入先前的节点。我们应该主动地启动 HDFS 平衡器来获得最优性能。平衡器的运行脚本为 bin/

start-balancer. sh，该脚本将在后台运行，直到集群达到平衡为止。管理员还可以提前终止它，即运行 bin/stop-balancer. sh。

当所有 DataNode 的利用率处于平均利用率加减一个阈值的范围内时，集群就被认为是平衡的。当启动一个平衡器脚本时，可以指定一个不同的阈值为 10%，当启动平衡器脚本时，也可以指定一个与此不同的阈值。例如，要设置阈值为 5% 以便让集群达到更优的均匀分布，需这样启动平衡器：bin/start-balancer. sh-threshold 5。

因为均衡操作会占用网络资源，建议在晚上或者周末做，此时集群可能不会太忙。或者，可以设置 dfs. balance. bandwidthPerSec 参数，以限制用于做均衡的带宽。

6.3　权限设置

HDFS 采用类似于 POSIX 语义的基本文件权限管理系统。每个文件有九种权限设置：与每个文件关联的所有者组和其他用户分别有读(r)、写(w)和执行(x)权限。不是所有的权限设置都有意义。在 HDFS 中不能执行文件，所以不设置文件的 x 权限。

目录的权限设置也严格遵循 POSIX 语义。权限 r 允许列出目录，权限 w 允许创建或删除文件或目录，权限 x 允许访问子目录。

当前版本的 HDFS 没有太多安全上的考虑。使用 HDFS 的权限管理系统只是为了防止在共享 Hadoop 集群的受信任用户之间发生数据的意外误用和覆盖。HDFS 不会对用户进行身份验证，并相信用户就是主机操作系统所声称的身份。Hadoop 的用户名就是登录名，等同于 whomi 所示的用户。组列表与 bash-c groups 所列出的一致。启动 name node 的用户名是一个例外，该用户名有一个特别的 Hadoop 用户名，即 superuser。这个超级用户可以执行任何文件操作，不受权限设置的约束。此外，管理员可以在一个超级用户组中通过配置参数 dfs. permissions. supergroup 来指定成员。所有超级用户组的成员也都是超级用户。

更改权限设置和所有权可以使用 bin/hadoop fs-chmod、-chown 以及-chgrp。使用方式与 UNIX 操作系统中对应的命令类似。

6.4　配额管理

默认情况下，HDFS 不设任何配额来限制一个目录中可以放多少内容。可以在指定目录上启用和指定所谓的名字配额，限制放在该目录下的文件名和目录名的个数。其主要用途是防止用户生成过多的小文件，令 NameNode 负担过重。以下命令用于设置和清除名字配额：

bin/hadoop dfsadmin-setQuota <N> directory [⋯directory]

% bin/hadoop dfsadmin-clrQuota directory [⋯directory]

HDFS 从 0. 19 版开始，还支持对每个目录做空间配额。它有助于管理一个用户或应用程序可用的存储量：

bin/hadoop dfsadmin-setSpaceQuota <N> directory [⋯directory]

% bin/hadoop dfsadmin-clrSpaceQuota directory [⋯directory]

在 SetSpaceQuota 命令中，代表每个目录配额的参数以字节为单位。这个参数还可以用后缀来表示单位。例如，20g 将代表 20GB，而 5t 则代表 5TB。所有的副本都计入配额。

若要获取目录的配额，以及它使用的名字个数和字节数，可使用带有-g 选项的 HDFS shell 命令 count：Bin/hadoop fs-count-q directory［…directory］。

6.5 启用回收站

除了文件权限之外，还有一个保护机制可以防止在 HDFS 上意外删除文件，这就是回收站。默认情况下该功能被禁用。当它启用后，用于删除文件的命令行程序不会立即删除文件。相反，它们暂时把文件移动到用户工作目录下的 . Trash/文件夹下。在用户设置的时间延迟到来之前，它都不会被永久删除。只要文件仍在 . Trash/文件夹下，你就可以通过将它移动到原来位置的方法来恢复它。

若要启用回收站功能并设置清空回收站的时间延迟，可以通过设置 core-site. Xml 的 fs. trash. interval 属性(以 min 为单位)。例如，如果希望用户有 24h(1440min)的时间来还原已删除的文件，应该在 core-site. xml 中设置：

```
< property >
< name > fs. trash. interval </name >
< value > 1440 </value >
</ property >
```

若将该值设置为 0，则将禁用回收站功能。

第 7 章　HDFS 实践

7.1　使用 HDFS 上传文件

7.1.1　实验目的

1）熟悉 HDFS 工作原理。
2）学会使用 HDFS 上传文件。

7.1.2　实验设备

1）硬件：云计算一体机、PC。
2）软件：SecureCRT。

7.1.3　实验内容

1）使用 SecureCRT 工具操作集群。
2）使用 HDFS 上传文件。

7.1.4　实验原理

程序源码如下：

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.URI;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.util.Progressable;
public class UploadFile2HDFS {
    /**
     * @param args
     * @throws IOException
```

```
    */
    public static void main(String[] args)throwsIOException {
        //TODO Auto-generated method stub
        if(args.length<2){
            System.out.println("parameter error");
        }else {
            String localSrc=args[0];
            String dst = args[1];
            InputStream in = new BufferedInputStream(new FileInputStream(
                    localSrc));
            Configuration conf=HadoopConfiguration.getConfiguration();
            FileSystem fs=FileSystem.get(URI.create(dst), conf);
            boolean overwrite=false;
            if(args.length >=3){
                overwrite=Boolean.parseBoolean(args[2]);
            }
            int bufferSize=4096;
            if(args.length>=4){
                bufferSize=Integer.parseInt(args[3]);
            }
            short replication=2;
            if(args.length>=5){
                replication=Short.parseShort(args[4]);
            }
            long blockSize=1024L*64*1024;
            if(args.length >=6){
                blockSize=1024L*Long.parseLong(args[5])*1024;
            }
            OutputStream out=fs.create(new Path(dst), overwrite, bufferSize,
                    replication,blockSize, new Progressable(){
                        public void progress(){
                            System.out.print(".");
                        }
                    });
            IOUtils.copyBytes(in, out, 4096, true);
        }
    }
}
```

7.1.5　实验步骤

1）先给本地机器上传输一个视频数据，如图 7-1 所示。

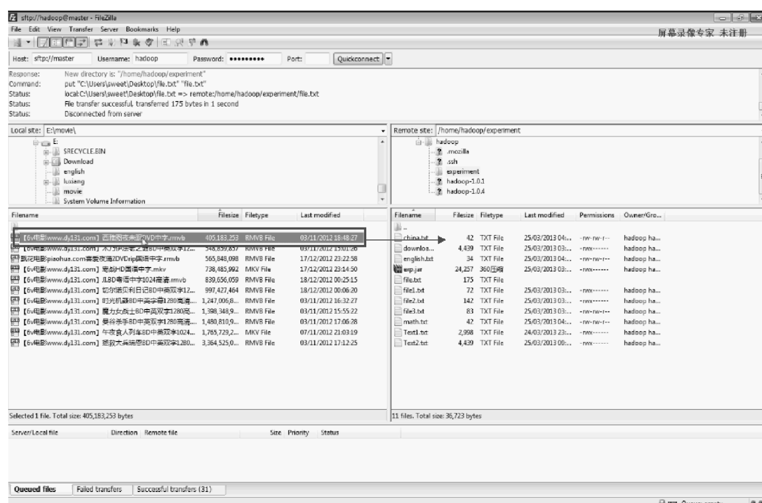


图 7-1　上传数据到集群

2）用命令行查看，文件传输是否成功 ll，如图 7-2 所示。

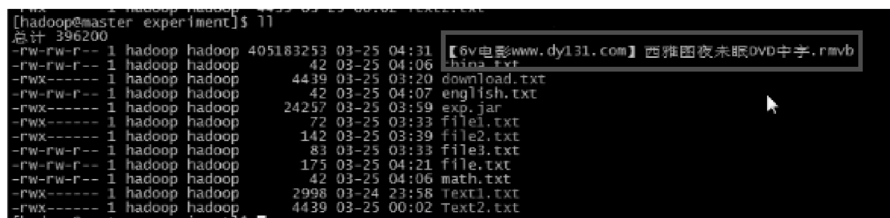


图 7-2　用命令行查看

3）因为名字比较长，所以用 mv 命令换一个名字，mv ＊. ＊，如图 7-3 所示。

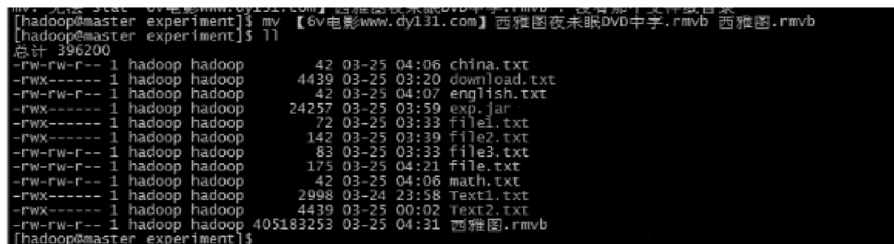


图 7-3　换一个名字

4）然后上传文件，其中有几个参数可以随意更改：

第一个是文件如果已经存在，是否要覆盖？true/false。

第二个是缓存大小，这个只有当文件非常大时会有用，4096 =4K。

第三个是是否要备份，备份几份，默认是 2。

第四个是块的大小，默认是 64MB，可更改。

5）然后执行程序 hadoop jar exp. jar UploadFile2HDFS＊. ＊/experiment/upload/＊. ＊false 4096 1 128。然后用页面访问查看，如图 7-4 所示。

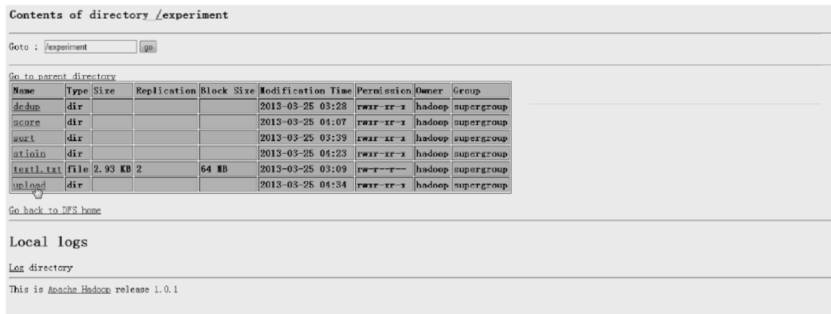


图 7-4　页面浏览上传文件夹

6）点击“upload”进入，如图 7-5 所示。

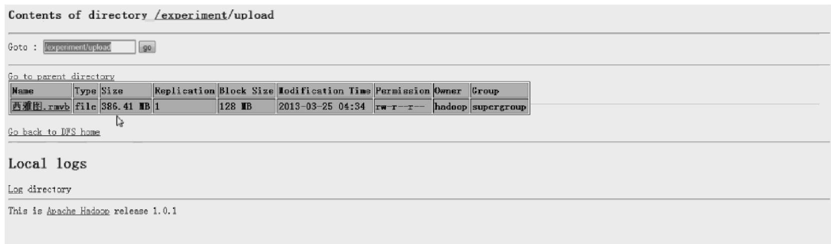


图 7-5　页面浏览上传文件

7）可以看到里面已经有了刚才上传的文件，并且看到了后面的参数。

7.2　使用 HDFS 浏览文件和目录

7.2.1　实验目的

1）熟悉 HDFS 浏览文件和目录的原理。
2）学会使用 HDFS 浏览文件和目录。

7.2.2　实验设备

1）硬件：云计算一体机、PC。
2）软件：SecureCRT。

7.2.3　实验内容

1）使用 SecureCRT 工具连接集群。
2）使用 HDFS 浏览文件和目录。

7.2.4　实验原理

文件查看、文件夹查看程序源码如下：

```
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

public class HDFSFileShow {

    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args)throws IOException {
        //TODO Auto-generated method stub
        if(args.length <1){
            System.out.println("parameter error");
        }else {
            SimpleDateFormat format = new SimpleDateFormat(
                    "yyyy-MM-dd HH:mm:ss");
            Configuration conf = HadoopConfiguration.getConfiguration();
            FileSystem hdfs = FileSystem.get(conf);
            Path path =new Path(args[0]);
            FileStatus stat = hdfs.getFileStatus(path);
            System.out.println("文件路径:"+stat.getPath().toUri().getPath());
            System.out.println("是否是目录:" + stat.isDir());
            System.out.println("文件大小:"+stat.getLen());
            System.out.println("修改日期:"
                    +format.format(new Date(stat.getModificationTime())));
            System.out.println("文件副本个数:"+stat.getReplication());
            System.out.println("文件拥有者:"+stat.getOwner());
            System.out.println("文件用户组:"+stat.getGroup());
            System.out.println("文件权限:"+stat.getPermission());
        }
    }
}
```

浏览目录程序源码如下：

```
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

public class HDFSFolderShow {

    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {
        //TODO Auto-generated method stub
        if (args.length < 1) {
            System.out.println("parameter error");
        } else {
            SimpleDateFormat format = new SimpleDateFormat(
                    "yyyy-MM-dd HH:mm:ss");
            Configuration conf = HadoopConfiguration.getConfiguration();
            FileSystem hdfs = FileSystem.get(conf);
            Path path = new Path(args[0]);
            FileStatus folder = hdfs.getFileStatus(path);
            if (folder.isDir()) {
                FileStatus[] array = hdfs.listStatus(path);
                for (FileStatus stat : array) {
                    System.out.println("文件路径:"
                            + stat.getPath().toUri().getPath());
                    System.out.println("是否是目录:" + stat.isDir());
                    System.out.println("文件大小:" + stat.getLen());
                    System.out
                            .println("修改日期:"
                                    + format.format(new Date(stat
                                            .getModificationTime())));
                   System.out.println("文件副本个数:" + stat.getReplication());
                    System.out.println("文件拥有者:" + stat.getOwner());
```

```
                        System.out.println ("文件用户组:" + stat.getGroup ());
                        System.out.println ("文件权限:" + stat.getPermission ());
                        System.out.println ("------------------------------------");
                    }
                } else {
                    System.out.println ("该文件为非目录!");
                }
            }
        }

    }
```

7.2.5　实验步骤

浏览文件或文件夹实验步骤如下：运行程序 hadoop jar exp.jar HDFSFileShow /experiment/upload/*.*可以直接看到文件的属性，如图 7-6 所示。

```
[hadoop@master experiment]$ hadoop jar exp.jar HDFSFileShow /experiment/upload/西雅图.rmvb
文件路径: /experiment/upload/西雅图.rmvb
是否是目录: false
文件大小: 405183253
修改日期: 2013-03-25 04:34:57
文件副本个数: 1
文件拥有者: hadoop
文件用户组: supergroup
文件权限: rw-r--r--
[hadoop@master experiment]$
```

图 7-6　浏览文件或文件夹

浏览目录实验步骤：运行程序，可以看到文件夹下的内容，hadoop jar exp.jar HDFSFolderShow /experiment/score，如图 7-7 所示。

```
[hadoop@master experiment]$ hadoop jar exp.jar HDFSFolderShow /experiment/score
文件路径: /experiment/score/china.txt
是否是目录: false
文件大小: 42
修改日期: 2013-03-25 04:06:25
文件副本个数: 2
文件拥有者: hadoop
文件用户组: supergroup
文件权限: rw-r--r--
------------------------------------
文件路径: /experiment/score/english.txt
是否是目录: false
文件大小: 42
修改日期: 2013-03-25 04:07:37
文件副本个数: 2
文件拥有者: hadoop
文件用户组: supergroup
文件权限: rw-r--r--
------------------------------------
文件路径: /experiment/score/math.txt
是否是目录: false
文件大小: 42
修改日期: 2013-03-25 04:06:23
文件副本个数: 2
文件拥有者: hadoop
文件用户组: supergroup
文件权限: rw-r--r--
------------------------------------
文件路径: /experiment/score/result
是否是目录: true
文件大小: 0
修改日期: 2013-03-25 04:08:30
文件副本个数: 0
文件拥有者: hadoop
文件用户组: supergroup
文件权限: rwxr-xr-x
------------------------------------
[hadoop@master experiment]$
```

图 7-7　浏览目录

7.3　使用 HDFS 打开、下载和删除文件

7.3.1　实验目的

1）熟悉 HDFS 打开、下载和删除文件的原理。
2）使用 HDFS 打开、下载和删除文件。

7.3.2　实验设备

1）硬件：云计算一体机、PC。
2）软件：SecureCRT。

7.3.3　实验内容

1）使用 SecureCRT 连接集群。
2）使用 HDFS 打开、下载和删除文件。

7.3.4　实验原理

打开文件程序源码如下：

```
import java.io.IOException;
import java.io.InputStream;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;

public class HDFSFileCat {

    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {
        //TODO Auto-generated method stub
        if(args.length != 1) {
            System.out.println("parameter error");
        }else{
            Configuration conf = HadoopConfiguration.getConfiguration();
            FileSystem hdfs = FileSystem.get(conf);
            Path path = new Path(args[0]);
```

```
                InputStream in  =  null;
                try{
                    in = hdfs. open( path) ;
                    IOUtils. copyBytes( in, System. out, 4096, false) ;
                }finally{
                    IOUtils. closeStream( in) ;
                }
            }
        }

}
```

下载文件程序源码如下：

```
import java. io. BufferedOutputStream;
import java. io. File;

import java. io. FileOutputStream;
import java. io. IOException;
import java. io. InputStream;
import java. io. OutputStream;
import org. apache. hadoop. conf. Configuration;
import org. apache. hadoop. fs. FileSystem;
import org. apache. hadoop. fs. Path;
import org. apache. hadoop. io. IOUtils;

public class DownloadHDFSFile {

    /* *
     * @ param args
     * @ throwsIOException
     */
    public static void main( String[ ] args) throwsIOException{
        //TODO Auto-generated method stub
        if( args. length <2) {
            System. out. println(" parameter error") ;
        }else{
            OutputStream bos = new BufferedOutputStream( new FileOutputStream(
                    new File( args[ 1 ] ) ) ) ;
            Configuration conf = HadoopConfiguration. getConfiguration( ) ;
```

```
                FileSystem fs = FileSystem.get(conf);
                Path hdfs = new Path(args[0]);
                InputStream hadopin = null;
                int bufferSize = 4096;
                if(args.length >= 3){
                    bufferSize = Integer.parseInt(args[2]);
                }
                try{
                    hadopin = fs.open(hdfs, bufferSize);
                    IOUtils.copyBytes(hadopin, bos, 4096, false);
                }finally{
                    IOUtils.closeStream(hadopin);
                    IOUtils.closeStream(bos);
                }
            }
        }
}
```

删除文件程序源码如下：

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

public class DeleteHDFSFile{

    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args)throwsIOException{
        //TODO Auto-generated method stub
        if(args.length!=1){
            System.out.println("parameter error");
        }else{
            Configuration conf = HadoopConfiguration.getConfiguration();
            FileSystem fs = FileSystem.get(conf);
            Path hdfs = new Path(args[0]);
            fs.delete(hdfs, true);
        }
```

```
        }
    }
```

7.3.5　实验步骤

1）打开文件，运行程序 hadoop jar exp. jar HDFSFileCat /experiment/score/math. txt，如图 7-8 所示。

```
[hadoop@master experiment]$ hadoop jar exp.jar HDFSFileCat /experiment/score/math.txt
张三 88
李四 99
王五 66
赵六 77[hadoop@master experiment]$
```

图 7-8　打开文件

2）下载文件，运行程序 hadoop jar exp. jar DownloadHDFSFile /experiment/score/english. txt . /download. txt。

3）查看是否下载下来 ll，如图 7-9 所示。

```
[hadoop@master experiment]$ ll
总计 396196
-rw-rw-r-- 1 hadoop hadoop        42 03-25 04:06 china.txt
-rwx------ 1 hadoop hadoop        42 03-25 04:41 download.txt
-rw-rw-r-- 1 hadoop hadoop        42 03-25 04:07 english.txt
-rwx------ 1 hadoop hadoop     24257 03-25 03:59 exp.jar
-rwx------ 1 hadoop hadoop        72 03-25 03:33 file1.txt
-rwx------ 1 hadoop hadoop       142 03-25 03:39 file2.txt
-rw-rw-r-- 1 hadoop hadoop        83 03-25 03:33 file3.txt
-rw-rw-r-- 1 hadoop hadoop       175 03-25 04:21 file.txt
-rw-rw-r-- 1 hadoop hadoop        42 03-25 04:06 math.txt
-rwx------ 1 hadoop hadoop      2998 03-24 23:58 Text1.txt
-rwx------ 1 hadoop hadoop      4439 03-25 00:02 Text2.txt
-rw-rw-r-- 1 hadoop hadoop 405183253 03-25 04:31 西雅图.rmvb
```

图 7-9　查看是否下载下来

4）再打开文件，看内容是否正确，cat download. txt，如图 7-10 所示。

```
[hadoop@master experiment]$ cat download.txt
张三 80
李四 70
王五 89
```

图 7-10　看内容是否正确

5）删除文件时，先查看要删除的文件，如图 7-11 所示。

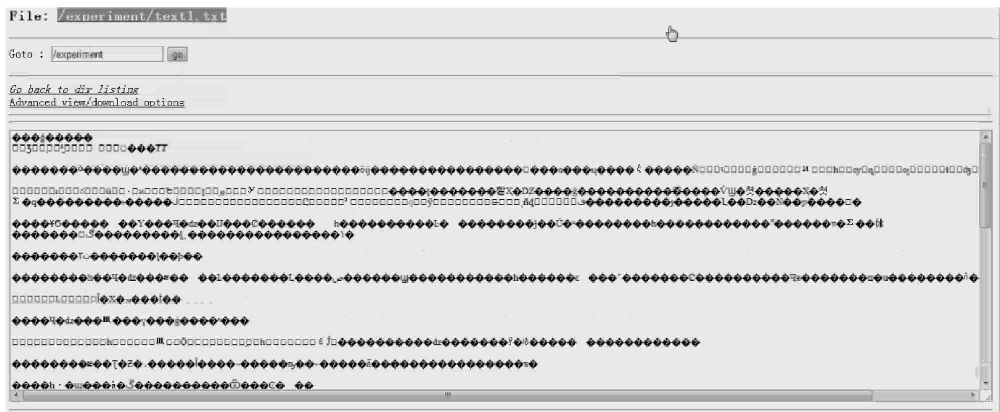


图 7-11　查看要删除的文件

6）运行程序 hadoop jar exp. jar DeleteHDFSFile /experiment/Text1. txt，然后刷新页面，如图 7-12 所示。



图 7-12　查看已经删除的文件

7）出错，说明文件已经不存在了，删除成功。

第 8 章　MapReduce 实践

8.1　数据去重实验

8.1.1　实验目的

“数据去重”主要是为了掌握和利用并行化思想来对数据进行有意义的筛选。统计大数据集上的数据种类个数、从网站日志中计算访问地等这些看似庞杂的任务都会涉及数据去重。下面就进入这个 MapReduce 程序设计实例。

8.1.2　实验设备

1）硬件：云计算一体机、PC。

2）软件：SecureCRT。

8.1.3　实验内容

对数据文件中的数据进行去重，数据文件中的每行都是一个数据。

样例输入如下所示：

1）文件一：

2012-3-1 a
2012-3-2 b
2012-3-3 c
2012-3-4 d
2012-3-5 a
2012-3-6 b
2012-3-7 c
2012-3-3 c

2）文件二：

2012-3-1 b
2012-3-2 a
2012-3-3 b
2012-3-4 d
2012-3-5 a
2012-3-6 c
2012-3-7 d
2012-3-3 c

样例输出如下所示：

2012-3-1 a
2012-3-1 b
2012-3-2 a
2012-3-2 b
2012-3-3 b
2012-3-3 c
2012-3-4 d
2012-3-5 a
2012-3-6 b
2012-3-6 c
2012-3-7 c
2012-3-7 d

8.1.4 实验原理

1. 设计思路

数据去重的最终目标是让原始数据中出现次数超过一次的数据在输出文件中只出现一次。我们自然而然会想到将同一个数据的所有记录都交给一台 reduce 机器，无论这个数据出现多少次，只要在最终结果中输出一次就可以了。具体就是 reduce 的输入应该以数据作为 key，而对 value-list 则没有要求。当 reduce 接收到一个 <key，value-list>时就直接将 key 复制到输出的 key 中，并将 value 设置成空值。

在 MapReduce 流程中，map 的输出 <key，value>经过 shuffle 过程聚集成 <key，value-list>后会交给 reduce。所以从设计好的 reduce 输入可以反推出 map 的输出 key 应为数据，value 任意。继续反推，map 输出数据的 key 为数据，而在这个实例中每个数据代表输入文件中的一行内容，所以 map 阶段要完成的任务就是在采用 Hadoop 默认的作业输入方式之后，将 value 设置为 key，并直接输出(输出中的 value 任意)。map 中的结果经过 shuffle 过程之后交给 reduce。reduce 阶段不会管每个 key 有多少个 value，它直接将输入的 key 复制为输出的 key，并输出就可以了(输出中的 value 被设置成空了)。

2. 程序源码

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```

```
import org. apache. hadoop. mapreduce. lib. output. FileOutputFormat;
public classDedup {

    //map 将输入中的 value 复制到输出数据的 key 上,并直接输出
    public static class Map extends Mapper < Object, Text, Text, Text > {
    private static Text line = new Text( );//每行数据
    //实现 map 函数
    public void map( Object key, Text value, Context context)
    throws IOException, InterruptedException {
    line = value;
    context. write( line, new Text("") ) ;
        }
    }

    //reduce 将输入中的 key 复制到输出数据的 key 上,并直接输出
    public static class Reduce extends Reducer < Text, Text, Text, Text > {
        //实现 reduce 函数

        public void reduce( Text key, Iterable < Text >  values, Context context)
                throws IOException, InterruptedException {
            context. write( key, new Text("") ) ;
        }
    }

    public static void main( String[ ]args) throws Exception {
        if( args. length!  =2 ) {
            System. err. println(" Usage: Data Deduplication  < in > < out >") ;
            System. exit(2) ;
        }
        Configuration conf = new Configuration( ) ;
        FileSystem fs = FileSystem. get( conf) ;
        Path result = new Path( args[ 1 ] ) ;
        if( fs. exists( result) ) {
            fs. delete( result, true) ;
        }
        Job job = new Job( conf, " Data Deduplication ") ;
        job. setJarByClass( Dedup. class) ;

        //设置 Map、Combine 和 Reduce 处理类
```

```
            job.setMapperClass(Map.class);
            job.setCombinerClass(Reduce.class);
            job.setReducerClass(Reduce.class);

            //设置输出类型
            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);

            //设置输入和输出目录
            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, result);
            System.exit(job.waitForCompletion(true) ? 0:1);
        }
    }
```

8.1.5　实验步骤

1)首先在/experiment/dedup/文件夹下上传两个实例文件，并且创建一个空文件，为了一会儿存放输出数据，如图 8-1 所示。

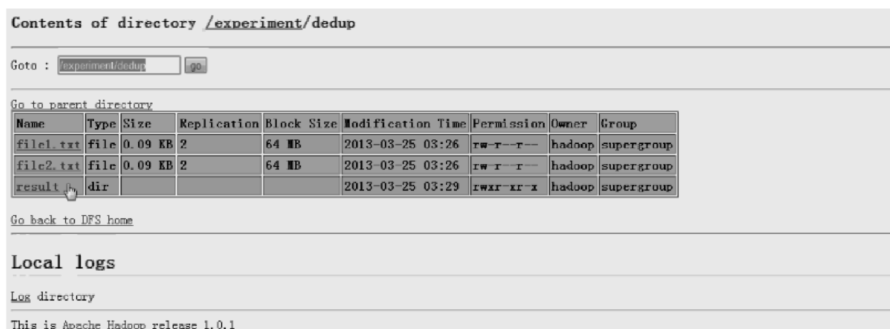


图 8-1　创建文件夹

2）然后执行程序 hadoop jar exp.jar Dedup /experiment/dedup /experiment/dedup/result，可以在页面下看到运行进度，运行完毕后，去 result 文件中看结果，如图 8-2 和图 8-3 所示。

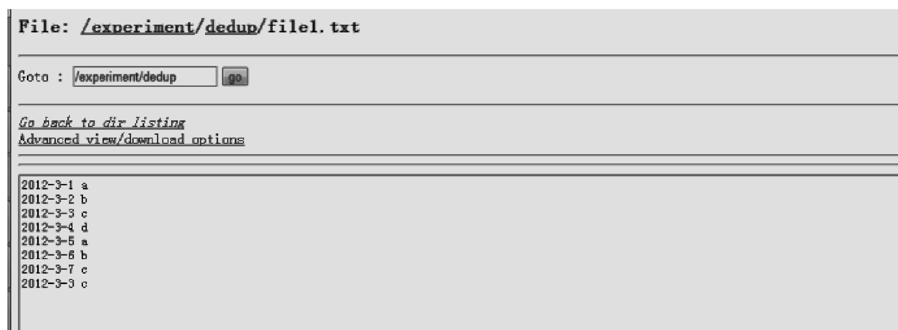


图 8-2　查看示例文件一

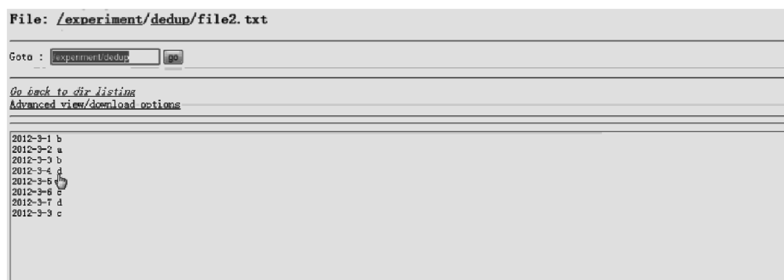


图 8-3　查看示例文件二

3）在页面能看到 Running Jobs 里有一个任务在运行。

4）打开这个任务，可以看到任务里面的内容，如图 8-4 所示。

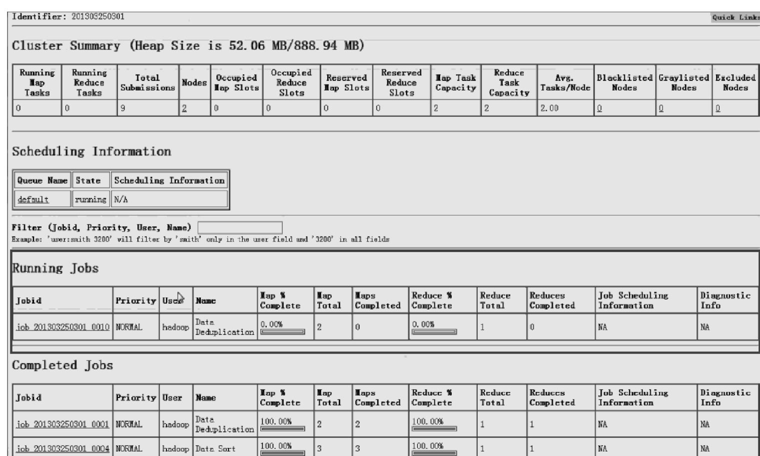


图 8-4　页面浏览任务

5）并且可以看到任务分配给了哪个主机去运行，如图 8-5 ~ 图 8-7 所示。

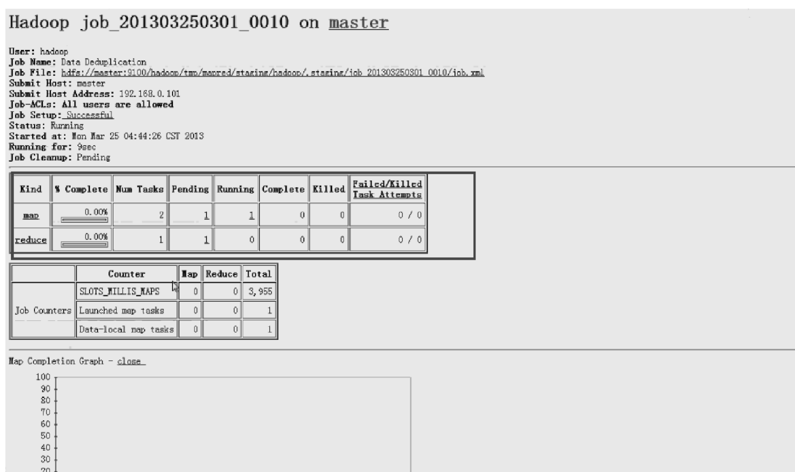


图 8-5　页面浏览执行中的任务

6）运行完毕后，打开 result 文件，点击 part-r-00000，就可以看到输出的数据，如图 8-8 和图 8-9 所示。

Hadoop map task list for job_201303250301_0010 on master

All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_201303250301_0010_m_000000	0.00%	initializing	25-三月-2013 04:44:33			0
task_201303250301_0010_m_000001	0.00%		25-三月-2013 04:44:36			0

Go back to JobTracker

This is Apache Hadoop release 1.0.1

图 8-6　页面浏览执行中的任务分配

Job job_201303250301_0010

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Finish Time	Errors	Task Logs	Counters	Actions
attempt_201303250301_0010_m_000000_0	/default-rack/slave2	SUCCEEDED	100.00%	29-三月-2013 01:14:52	29-三月-2013 01:14:55 (3sec)		Last 4KB Last 8KB All	15	

Input Split Locations

/default-rack/slave2
/default-rack/slave1

Go back to the job
Go back to JobTracker

This is Apache Hadoop release 1.0.1

图 8-7　任务详情

Contents of directory /experiment/dedup/result

Goto : /experiment/dedup/result go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 KB	2	64 MB	2013-03-25 04:44	rw-r--r--	hadoop	supergroup
_logs	dir				2013-03-25 04:44	rwxr-xr-x	hadoop	supergroup
part-r-00000	file	0.14 KB	2	64 MB	2013-03-25 04:44	rw-r--r--	hadoop	supergroup

Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.0.1

图 8-8　查看输出数据

File: /experiment/dedup/result/part-r-00000

Goto : /experiment/dedup/result go

Go back to dir listing
Advanced view/download options

```
2012-3-1 a
2012-3-1 b
2012-3-2 a
2012-3-2 b
2012-3-3 b
2012-3-3 c
2012-3-4 d
2012-3-5 a
2012-3-6 b
2012-3-6 c
2012-3-7 c
2012-3-7 d
```

Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768 Refresh

Total number of blocks: 1
-8925127324055411115: 192.168.0.102:50010 192.168.0.103:50010

Go back to DFS home

Local logs

图 8-9　查看输出数据内容

7）进行核对，发现已经去重成功。

8.2　数据排序实验

8.2.1　实验目的

“数据排序”是许多实际任务执行时要完成的第一项工作，比如学生成绩评比、数据建立索引等。这个实例和数据去重类似，都是先对原始数据进行初步处理，为进一步的数据操作打好基础。下面进入这个实例。

8.2.2　实验设备

1）硬件：云计算一体机、PC。

2）软件：SecureCRT。

8.2.3　实验内容

1. 实例描述

对输入文件中的数据进行排序。输入文件中的每行内容均为一个数字，即一个数据。要求在输出中每行有两个间隔的数字，其中，第一个代表原始数据在原始数据集中的位次，第二个代表原始数据。

2. 样例输入

1）文件一：如图 8-10 所示。



图 8-10　查看样例文件一

2）文件二：如图 8-11 所示。

3）文件三：如图 8-12 所示。

8.2.4　实验原理

1. 设计思路

这个实例仅仅要求对输入数据进行排序，熟悉 MapReduce 过程的读者会很快想到在 MapReduce 过程中就有排序，是否可以利用这个默认的排序，而不需要自己再实现具体的排序



图 8-11　查看样例文件二



图 8-12　查看样例文件三

呢？答案是肯定的。

但是在使用之前首先需要了解它的默认排序规则。它是按照 key 值进行排序的，如果 key 为封装 int 的 IntWritable 类型，那么 MapReduce 按照数字大小对 key 排序，如果 key 为封装为 String 的 Text 类型，那么 MapReduce 按照字典顺序对字符串排序。

了解了这个细节，我们就知道了应该使用封装 int 的 IntWritable 型数据结构。也就是在 map 中将读入的数据转化成 IntWritable 型，然后作为 key 值输出(value 任意)。reduce 拿到 <key，value-list>之后，将输入的 key 作为 value 输出，并根据 value-list 中元素的个数决定输出的次数。输出的 key(即代码中的 linenum)是一个全局变量，它统计当前 key 的位次。需要注意的是这个程序中没有配置 Combiner，也就是在 MapReduce 过程中不使用 Combiner。这主要是因为使用 map 和 reduce 就已经能够完成任务了。

2. 程序源码

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Sort {

    //map 将输入中的 value 化成 IntWritable 类型,作为输出的 key
    public static class Map extends
            Mapper<Object, Text, IntWritable, IntWritable> {
        private static IntWritable data = new IntWritable();

        //实现 map 函数
        public void map(Object key,Text value,Context context)
                throws IOException,InterruptedException {
            String line = value.toString();
            if(line! =null&&! "".equals(line)){
                data.set(Integer.parseInt(line));
                context.write(data,new IntWritable(1));
            }
        }
    }

    //reduce 将输入中的 key 复制到输出数据的 key 上,
    //然后根据输入的 value - list 中元素的个数决定 key 的输出次数
    //用全局 linenum 来代表 key 的位次
    public static class Reduce extends
            Reducer<IntWritable,IntWritable,IntWritable,IntWritable>{
        private static IntWritable linenum = new IntWritable(1);

        //实现 reduce 函数
        public void reduce(IntWritable key,Iterable<IntWritable> values,
                Context context) throws IOException, InterruptedException{
            for(IntWritable val:values){
                context.write(linenum,key);
```

```
                linenum = new IntWritable(linenum.get() + 1);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        if(args.length != 2) {
            System.err.println("Usage: Data Deduplication <in> <out>");
            System.exit(2);
        }
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);
        Path result = new Path(args[1]);
        if(fs.exists(result)) {
            fs.delete(result, true);
        }
        Job job = new Job(conf, "Data Sort");
        job.setJarByClass(Sort.class);

        //设置 Map 和 Reduce 处理类
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

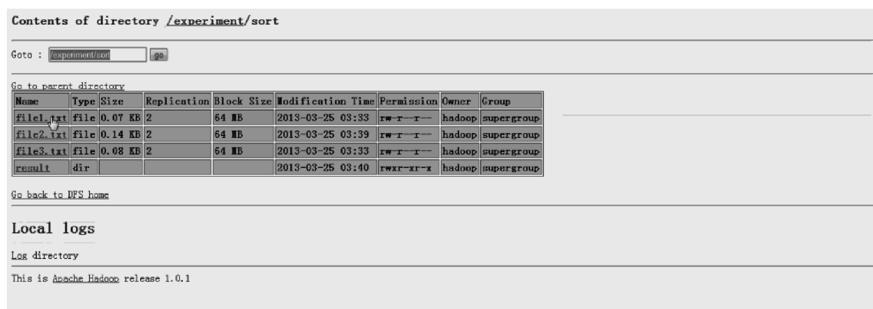
        //设置输出类型
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(IntWritable.class);

        //设置输入和输出目录
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, result);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

8.2.5　实验步骤

1）首先在文件夹/experiment/sort/下上传三个实例文件，并且创建一个空文件，为了一会儿存放输出数据，如图 8-13 所示。

2）然后执行程序 hadoop jar exp.jar Sort /experiment/sort /experiment/sort/result，在页面下查看进度，如图 8-14 和图 8-15 所示。



Contents of directory /experiment/sort

Goto : /experiment/sort go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
file1.txt	file	0.07 KB	2	64 MB	2013-03-25 03:33	rw-r--r--	hadoop	supergroup
file2.txt	file	0.14 KB	2	64 MB	2013-03-25 03:39	rw-r--r--	hadoop	supergroup
file3.txt	file	0.08 KB	2	64 MB	2013-03-25 03:33	rw-r--r--	hadoop	supergroup
result	dir				2013-03-25 03:40	rwxr-xr-x	hadoop	supergroup

Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.0.1

图 8-13　创建文件夹

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201303250301_0011	NORMAL	hadoop	Data Sort	0.00%	3	0	0.00%	1	0	NA	NA

图 8-14　查看正在运行的任务

Hadoop job_201303250301_0011 on master

User: hadoop
Job Name: Data Sort
Job File: hdfs://master:9100/hadoop/tmp/mapred/staging/hadoop/.staging/job_201303250301_0011/job.xml
Submit Host: master
Submit Host Address: 192.168.0.101
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Running
Started at: Mon Mar 25 04:48:03 CST 2013
Running for: 8sec
Job Cleanup: Pending

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	0.00%	3	1	2	0	0	0 / 0
reduce	0.00%	1	1	0	0	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	SLOTS_MILLIS_MAPS	0	0	3,980
	Launched map tasks	0	0	2
	Data-local map tasks	0	0	2

图 8-15　查看任务进度

3）可以看到任务分配给谁去执行，如图 8-16 ~ 图 8-18 所示。

Hadoop map task list for job_201303250301_0011 on master

All Tasks

Task	Complete	Status	Start Time	Finish Time	Errors	Counters
task_201303250301_0011_m_000000	0.00%	initializing	25-三月-2013 04:48:09			0
task_201303250301_0011_m_000001	0.00%	initializing	25-三月-2013 04:48:09			0
task_201303250301_0011_m_000002	0.00%					0

Go back to JobTracker

This is Apache Hadoop release 1.0.1

图 8-16　查看任务分配

4）完成后可以看到前面是序号，后面是数据，已经排序成功，如图 8-19 所示。

5）点击 part-r-00000 看数据，如图 8-20 所示。

Job job_201303250301_0011

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Finish Time	Errors	Task Logs	Counters	Actions
attempt_201303250301_0011_m_000000_0	/default-rack/slave1	SUCCEEDED	100.00%	29-三月-2013 01:18:20	29-三月-2013 01:18:23 (3sec)		Last 4KB Last 8KB All	15	

Input Split Locations

/default-rack/slave2
/default-rack/slave1

Go back to the job
Go back to JobTracker

This is Apache Hadoop release 1.0.1

图 8-17　查看任务详情一

Job job_201303250301_0011

All Task Attempts

Task Attempts	Machine	Status	Progress	Start Time	Finish Time	Errors	Task Logs	Counters	Actions
attempt_201303250301_0011_m_000001_0	/default-rack/slave2	SUCCEEDED	100.00%	29-三月-2013 01:18:28	29-三月-2013 01:18:32 (3sec)		Last 4KB Last 8KB All	15	

Input Split Locations

/default-rack/slave2
/default-rack/slave1

Go back to the job
Go back to JobTracker

This is Apache Hadoop release 1.0.1

图 8-18　查看任务详情二

Contents of directory /experiment/sort/result

Goto : /experiment/sort/result go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 KB	2	64 MB	2013-03-25 04:48	rw-r--r--	hadoop	supergroup
_logs	dir				2013-03-25 04:48	rwxr-xr-x	hadoop	supergroup
part-r-00000	file	0.43 KB	2	64 MB	2013-03-25 04:48	rw-r--r--	hadoop	supergroup

Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.0.1

图 8-19　查看输出结果文件

File: /experiment/sort/result/part-r-00000

Goto : /experiment/sort/result go

Go back to dir listing
Advanced view/download options

```
1	2
2	2
3	2
4	2
5	2
6	3
7	3
8	4
9	4
10	4
11	4
12	5
13	5
14	5
15	6
16	6
17	7
18	7
19	7
20	7
21	7
22	8
23	8
24	12
25	23
```

Download this file
Tail this file
Chunk size to view (in bytes, up to file's DFS block size): 32768 Refresh

图 8-20　查看输出结果文件内容

8.3　平均成绩实验

8.3.1　实验目的

“平均成绩”主要目的还是在重温经典的“WordCount”例子，可以说是在此基础上的微变化版，该实例主要就是实现一个计算学生平均成绩的例子。

8.3.2　实验设备

1）硬件：云计算一体机、PC。

2）软件：SecureCRT。

8.3.3　实验内容

对输入文件中的数据进行计算，得出学生的平均成绩。输入文件中的每行内容均为一个学生的姓名和他相应的成绩，如果有多门学科，则每门学科为一个文件。要求在输出中每行有两个间隔的数据，其中，第一个代表学生的姓名，第二个代表其平均成绩。

1. 样本输入

1）数学：

张三　　88
李四　　99
王五　　66
赵六　　77

2）语文：

张三　　78
李四　　89
王五　　96
赵六　　67

3）英语：

张三　　80
李四　　82
王五　　84
赵六　　86

2. 样本输出

张三　　82
李四　　90
王五　　82
赵六　　76

8.3.4　实验原理

1. 设计思路

计算学生平均成绩是一个仿“WordCount”例子，用来重温一下开发 MapReduce 程序的流程。程序包括两部分的内容：Map 部分和 Reduce 部分，分别实现了 map 和 reduce 的功能。

Map 处理的是一个纯文本文件，文件中存放的数据是每一行表示一个学生的姓名和他相应的一科成绩。Mapper 处理的数据是由 InputFormat 分解过的数据集，其中 InputFormat 的作用是将数据集切割成小数据集 InputSplit，每一个 InputSlit 将由一个 Mapper 负责处理。此外，InputFormat 中还提供了一个 RecordReader 的实现，并将一个 InputSplit 解析成 <key，value> 对提供给了 map 函数。InputFormat 的默认值是 TextInputFormat，它针对文本文件，按行将文本切割成 InputSlit，并用 LineRecordReader 将 InputSplit 解析成 <key，value> 对，key 是行在文本中的位置，value 是文件中的一行。

Map 的结果会通过 partion 分发到 Reducer，Reducer 做完 Reduce 操作后，将通过以格式 OutputFormat 输出。

Mapper 最终处理的结果 <key，value> 对，会送到 Reducer 中进行合并，合并的时候，有相同 key 的键/值对则送到同一个 Reducer 上。Reducer 是所有用户定制 Reducer 类的基础，它的输入是 key 和这个 key 对应的所有 value 的一个迭代器，同时还有 Reducer 的上下文。Reduce 的结果由 Reducer. Context 的 write 方法输出到文件中。

2. 程序源码

```
import java.io.IOException;
import java.util.Iterator;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Score {
```

```
public static class Map extends
        Mapper < LongWritable, Text, Text, IntWritable > {
    //实现 map 函数
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        //将输入的纯文本文件的数据转化成 String
        String line = value.toString();
        //将输入的数据首先按行进行分割
        StringTokenizer tokenizerArticle = new StringTokenizer(line, "\n");
        //分别对每一行进行处理
        while(tokenizerArticle.hasMoreElements()){
            //每行按空格划分
            StringTokenizer tokenizerLine = new StringTokenizer(
                    tokenizerArticle.nextToken());
            String strName = tokenizerLine.nextToken();//学生姓名部分
            String strScore = tokenizerLine.nextToken();//成绩部分
            Text name = new Text(strName);
            int scoreInt = Integer.parseInt(strScore);
            //输出姓名和成绩
            context.write(name, new IntWritable(scoreInt));
        }
    }
}

public static class Reduce extends
        Reducer < Text, IntWritable, Text, IntWritable > {
    //实现 reduce 函数
    public void reduce(Text key, Iterable < IntWritable > values,
            Context context)throws IOException, InterruptedException {
        int sum = 0;
        int count = 0;
        Iterator < IntWritable > iterator = values.iterator();
        while(iterator.hasNext()){
            sum + = iterator.next().get();//计算总分
            count + +;//统计总的科目数
        }
        int average = (int)sum/count;//计算平均成绩
        context.write(key, new IntWritable(average));
```

```
        }
    }

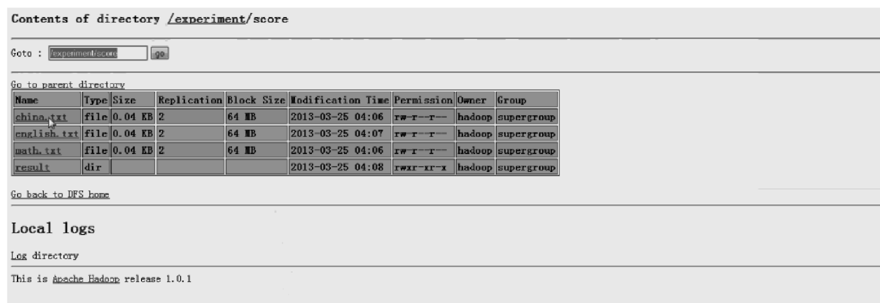
    public static void main(String[] args)throws Exception {
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);
        Path result = new Path(args[1]);
        if(fs.exists(result)){
            fs.delete(result, true);
        }
        if(args.length! =2){
            System.err.println("Usage: Score Average <in> <out>");
            System.exit(2);
        }
        Job job = new Job(conf, "Score Average");
        job.setJarByClass(Score.class);
        //设置 Map、Combine 和 Reduce 处理类
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);
        //设置输出类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        //将输入的数据集分割成小数据块 splites,提供一个 RecordReder 的实现
        job.setInputFormatClass(TextInputFormat.class);
        //提供一个 RecordWriter 的实现,负责数据输出
        job.setOutputFormatClass(TextOutputFormat.class);
        //设置输入和输出目录
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, result);
        System.exit(job.waitForCompletion(true)? 0:1);
    }
}
```

8.3.5　实验步骤

1）首先把数据上传到/experiment/score/文件夹下，并创建一个空文件为了存放输出的数据，如图 8-21 所示。

2）然后执行程序 hadoop jar exp.jar Score /experiment/score /experiment/score/result，在



Contents of directory /experiment/score

Goto : /experiment/score go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
china.txt	file	0.04 KB	2	64 MB	2013-03-25 04:06	rw-r--r--	hadoop	supergroup
english.txt	file	0.04 KB	2	64 MB	2013-03-25 04:07	rw-r--r--	hadoop	supergroup
math.txt	file	0.04 KB	2	64 MB	2013-03-25 04:06	rw-r--r--	hadoop	supergroup
result	dir				2013-03-25 04:08	rwxr-xr-x	hadoop	supergroup

Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.0.1

图 8-21　创建文件夹

页面下看程序的运行进度，如图 8-22 和图 8-23 所示。

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201303250301_0012	NORMAL	hadoop	Score Average	0.00%	3	0	0.00%	1	0	NA	NA

Completed Jobs

图 8-22　查看正在运行的任务

Hadoop job_201303250301_0012 on master

User: hadoop
Job Name: Score Average
Job File: hdfs://master:9100/hadoop/tmp/mapred/staging/hadoop/.staging/job_201303250301_0012/job.xml
Submit Host: master
Submit Host Address: 192.168.0.101
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Running
Started at: Mon Mar 25 04:51:09 CST 2013
Running for: 14sec
Job Cleanup: Pending

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	66.66%	3	0	1	2	0	0 / 0
reduce	0.00%	1	0	1	0	0	0 / 0

	Counter	Map	Reduce	Total
Job Counters	SLOTS_MILLIS_MAPS	0	0	11,175
	Launched reduce tasks	0	0	1
	Launched map tasks	[illegible]	0	3
	Data-local map tasks	0	0	3
File Input Format Counters	Bytes Read	84	0	84
FileSystemCounters	HDFS_BYTES_READ	306	0	306
	FILE_BYTES_WRITTEN	43,626	0	43,626
	Map output materialized bytes	116	0	116
	Combine output records	8	0	8
	Map input records	8	0	8

图 8-23　查看任务进程

3）完成后查看输出结果，已经得到平均成绩，实验成功，如图 8-24 和图 8-25 所示。

Contents of directory /experiment/score/result

Goto : /experiment/score/result go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
_SUCCESS	file	0 KB	2	64 MB	2013-03-25 04:51	rw-r--r--	hadoop	supergroup
_logs	dir				2013-03-25 04:51	rwxr-xr-x	hadoop	supergroup
part-r-00000	file	0.04 KB	2	64 MB	2013-03-25 04:51	rw-r--r--	hadoop	supergroup

Go back to DFS home

Local logs

Log directory

This is Apache Hadoop release 1.0.1

图 8-24　查看输出文件

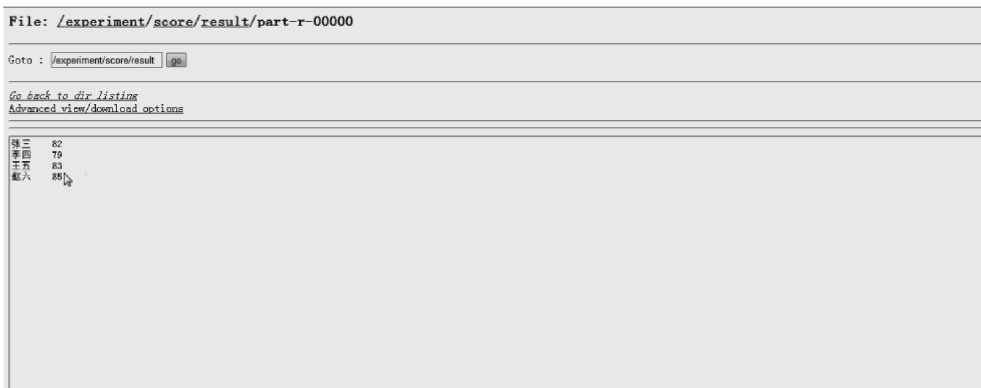


图 8-25　查看输出文件内容

8.4　单表关联实验

8.4.1　实验目的

前面的实例都是在数据上进行一些简单的处理，为进一步的操作打基础。“单表关联”这个实例要求从给出的数据中寻找所关心的数据，它是对原始数据所包含信息的挖掘。下面进入这个实例。

8.4.2　实验设备

1）硬件：云计算一体机、PC。

2）软件：SecureCRT。

8.4.3　实验内容

实例中给出 child-parent（孩子—父母）表，要求输出 grandchild-grandparent（孙子—爷奶）表。

样例输入如下所示：

文件：

```
child   parent
Tom     Lucy
Tom     Jack
Jone    Lucy
Jone    Jack
Lucy        Mary
Lucy        Ben
Jack    Alice
Jack    Jesse
```

Terry　　Alice
Terry　　Jesse
Philip　　Terry
Philip　　Alma
Mark　　Terry
Mark　　Alma

家族树状关系谱如图 8-26 所示。

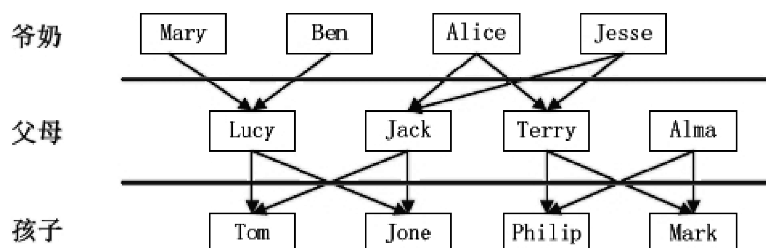


图 8-26　家族关系谱

样例输出如下所示：

文件：

grandchild　　grandparent
Tom　　Alice
Tom　　Jesse
Jone　　Alice
Jone　　Jesse
Tom　　Mary
Tom　　Ben
Jone　　Mary
Jone　　Ben
Philip　　Alice
Philip　　Jesse
Mark　　Alice
Mark　　Jesse

8.4.4　实验原理

1. 设计思路

分析这个实例，显然需要进行单表连接，连接的是左表的 parent 列和右表的 child 列，且左表和右表是同一个表。

连接结果中除去连接的两列就是所需要的结果——“grandchild--grandparent”表。要用 MapReduce 解决这个实例，首先应该考虑如何实现表的自连接；其次就是连接列的设置；最后是结果的整理。

考虑到 MapReduce 的 shuffle 过程会将相同的 key 会连接在一起，所以可以将 map 结果的 key 设置成待连接的列，然后列中相同的值就自然会连接在一起了。再与最开始的分析联系起来。

要连接的是左表的 parent 列和右表的 child 列，且左表和右表是同一个表，所以在 map 阶段将读入数据分割成 child 和 parent 之后，会将 parent 设置成 key，child 设置成 value 输出，并作为左表；再将同一对 child 和 parent 中的 child 设置成 key，parent 设置成 value 进行输出，作为右表。为了区分输出中的左、右表，需要在输出的 value 中再加上左右表的信息，比如在 value 的 String 最开始处加上字符 1 表示左表，加上字符 2 表示右表。这样在 map 的结果中就形成了左表和右表，然后在 shuffle 过程中完成连接。reduce 接收到连接的结果，其中每个 key 的 value-list 就包含了"grandchild--grandparent" 关系。取出每个 key 的 value-list 进行解析，将左表中的 child 放入一个数组，右表中的 parent 放入一个数组，然后对两个数组求笛卡尔积就是最后的结果了。

2. 程序源码

```
import java. io. IOException;
import java. util. * ;

import org. apache. hadoop. conf. Configuration;
import org. apache. hadoop. fs. FileSystem;
import org. apache. hadoop. fs. Path;
import org. apache. hadoop. io. Text;
import org. apache. hadoop. mapreduce. Job;
import org. apache. hadoop. mapreduce. Mapper;
import org. apache. hadoop. mapreduce. Reducer;
import org. apache. hadoop. mapreduce. lib. input. FileInputFormat;
import org. apache. hadoop. mapreduce. lib. output. FileOutputFormat;

public classSTjoin {

    public static int time =0;

    / *
      * map 将输出分割 child 和 parent,然后正序输出一次作为右表，反序输出一次作为
左表,需要注意的是在输出的 value 中必须加上左右表的区别标识
      * /
    public static class Map extends Mapper < Object, Text, Text, Text > {
    //实现 map 函数
    public void map( Object key, Text value, Context context)
                throws IOException, InterruptedException{
```

```
            String childname = new String();//孩子名称
            String parentname = new String();//父母名称
            String relationtype = new String();//左右表标识

            //输入的一行预处理文本
            StringTokenizer itr = new StringTokenizer(value.toString());
            String[] values = new String[2];
            int i = 0;
            while(itr.hasMoreTokens()){
                values[i] = itr.nextToken();
                i++;
            }

            if(values[0].compareTo("child")! =0){
                childname = values[0];
                parentname = values[1];
                //输出左表
                relationtype = "1";
                context.write(new Text(values[1]),new Text(relationtype + "+"
                        + childname + "+" + parentname));
                //输出右表
                relationtype = "2";
                context.write(new Text(values[0]),new Text(relationtype + "+"
                        + childname + "+" + parentname));
            }
        }
    }

    public static class Reduce extends Reducer<Text,Text,Text,Text>{
        //实现 reduce 函数
        public void reduce(Text key,Iterable<Text> values, Context context)
                throws IOException, InterruptedException{
            //输出表头
            if(0 == time){
                context.write(new Text("grandchild"),new Text("grandparent"));
                time++;
            }
            int grandchildnum = 0;
```

```
String[ ] grandchild = new String[10];
int grandparentnum = 0;
String[ ] grandparent = new String[10];

Iterator ite = values. iterator( );
while( ite. hasNext( ) ){
    String record = ite. next( ). toString( );
    int len = record. length( );
    int i = 2;
    if(0 = = len){
        continue;
    }

    //取得左、右表标识
    char relationtype = record. charAt(0);
    //定义孩子和父母变量
    String childname = new String( );
    String parentname = new String( );

    //获取 value - list 中 value 的 child
    while( record. charAt( i)! ='+'){
        childname + = record. charAt( i);
        i + +;
    }
    i = i + 1;
    //获取 value - list 中 value 的 parent
    while( i < len){
        parentname + = record. charAt( i);
        i + +;
    }

    //左表,取出 child 放入 grandchildren
    if('1'= = relationtype){
        grandchild[ grandchildnum] = childname;
        grandchildnum + +;
    }
    //右表,取出 parent 放入 grandparent
    if('2'= = relationtype){
        grandparent[ grandparentnum] = parentname;
```

```
                    grandparentnum++;
                }
            }

            //grandchild 和 grandparent 数组求笛卡尔积
            if(0!=grandchildnum&&0!=grandparentnum){
                for(int m=0;m<grandchildnum;m++){
                    for(int n=0;n<grandparentnum;n++){
                        //输出结果
                        context.write(new Text(grandchild[m]),new Text(
                                grandparent[n]));
                    }
                }
            }
        }
    }

    public static void main(String[] args)throws Exception{
        Configuration conf=new Configuration();
        FileSystem fs=FileSystem.get(conf);
        Path result=new Path(args[1]);
        if(fs.exists(result)){
            fs.delete(result, true);
        }
        if(args.length!=2){
            System.err.println("Usage: Single Table Join <in> <out>");
            System.exit(2);
        }

        Job job=new Job(conf,"Single Table Join");
        job.setJarByClass(STjoin.class);

        //设置 Map 和 Reduce 处理类
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);

        //设置输出类型
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
```

```
        //设置输入和输出目录
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, result);
        System.exit(job.waitForCompletion(true)? 0:1);
    }
}
```

8.4.5　实验步骤

1. 代码结果

1）首先上传事例文件到/experiment/st. join/文件夹下，并且创建一个空文件，方便存放输出数据。

2）运行程序 hadoop jar exp. jar STjoin /experiment/stjoin /experiment/stjoin/result，在页面看到进度，如图 8-27 和图 8-28 所示。

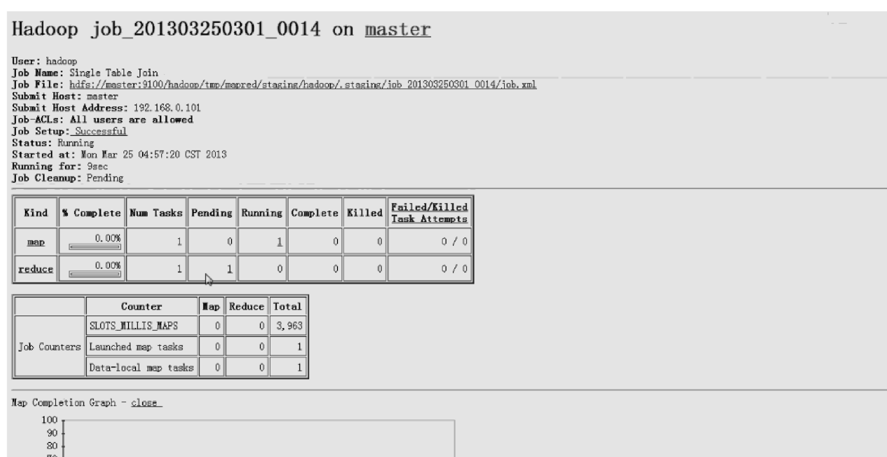


图 8-27　查看运行任务

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201303250301_0014	NORMAL	hadoop	Single Table Join	0.00%	1	0	0.00%	1	0	NA	NA

Completed Jobs

图 8-28　查看任务进度

2. 运行详解

（1） map 处理

map 函数输出结果如图 8-29 所示。

（2） shuffle 处理

在 shuffle 过程中完成连接，见表 8-1 ~ 表 8-3。

child	parent	→→	忽略此行
Tom	Lucy	→→	<Lucy，1+Tom+Lucy> <Tom，2+Tom+Lucy>
Tom	Jack	→→	<Jack，1+Tom+Jack> <Tom，2+Tom+Jack>
Jone	Lucy	→→	<Lucy，1+Jone+Lucy> <Jone，2+Jone+Lucy>
Jone	Jack	→→	<Jack，1+Jone+Jack> <Jone，2+Jone+Jack>
Lucy	Mary	→→	<Mary，1+Lucy+Mary> <Lucy，2+Lucy+Mary>
Lucy	Ben	→→	<Ben，1+Lucy+Ben> <Lucy，2+Lucy+Ben>
Jack	Alice	→→	<Alice，1+Jack+Alice> <Jack，2+Jack+Alice>
Jack	Jesse	→→	<Jesse，1+Jack+Jesse> <Jack，2+Jack+Jesse>
Terry	Alice	→→	<Alice，1+Terry+Alice> <Terry，2+Terry+Alice>
Terry	Jesse	→→	<Jesse，1+Terry+Jesse> <Terry，2+Terry+Jesse>
Philip	Terry	→→	<Terry，1+Philip+Terry> <Philip，2+Philip+Terry>
Philip	Alma	→→	<Alma，1+Philip+Alma> <Philip，2+Philip+Alma>
Mark	Terry	→→	<Terry，1+Mark+Terry> <Mark，2+Mark+Terry>
Mark	Alma	→→	<Alma，1+Mark+Alma> <Mark，2+Mark+Alma>

图 8-29　map 函数输出结果

表 8-1　shuffle 连接

map 函数输出	排序结果	shuffle 连接
< Lucy，1 + Tom + Lucy >	< Alice，1 + Jack + Alice >	< Alice，1 + Jack + Alice，
< Tom，2 + Tom + Lucy >	< Alice，1 + Terry + Alice >	1 + Terry + Alice ，
< Jack，1 + Tom + Jack >	< Alma，1 + Philip + Alma >	1 + Philip + Alma，
< Tom，2 + Tom + Jack >	< Alma，1 + Mark + Alma >	1 + Mark + Alma >
< Lucy，1 + Jone + Lucy >	< Ben，1 + Lucy + Ben >	< Ben，1 + Lucy + Ben >
< Jone，2 + Jone + Lucy >	< Jack，1 + Tom + Jack >	< Jack，1 + Tom + Jack，

表 8-2　无效及有效的 shuffle 连接

无效的 shuffle 连接	有效的 shuffle 连接
< Alice，1 + Jack + Alice， 1 + Terry + Alice ， 1 + Philip + Alma， 1 + Mark + Alma >	< Jack，1 + Tom + Jack， 1 + Jone + Jack，

（续）

无效的 shuffle 连接	有效的 shuffle 连接
< Ben，1 + Lucy + Ben > < Jesse，1 + Jack + Jesse， 1 + Terry + Jesse > < Jone，2 + Jone + Lucy， 2 + Jone + Jack > < Mary，1 + Lucy + Mary， 2 + Mark + Terry， 2 + Mark + Alma > < Philip，2 + Philip + Terry， 2 + Philip + Alma > < Tom，2 + Tom + Lucy， 2 + Tom + Jack >	2 + Jack + Alice， 2 + Jack + Jesse > < Lucy，1 + Tom + Lucy， 1 + Jone + Lucy， 2 + Lucy + Mary， 2 + Lucy + Ben > < Terry，2 + Terry + Alice， 2 + Terry + Jesse， 1 + Philip + Terry， 1 + Mark + Terry >

表 8-3　shuffle 连接过程

< Jack，1 + Jone + Jack >	< Jack，1 + Jone + Jack >	1 + Jone + Jack，
< Jone，2 + Jone + Jack >	< Jack，2 + Jack + Alice >	2 + Jack + Alice，
< Mary，1 + Lucy + Mary >	< Jack，2 + Jack + Jesse >	2 + Jack + Jesse >
< Lucy，2 + Lucy + Mary >	< Jesse，1 + Jack + Jesse >	< Jesse，1 + Jack + Jesse，
< Ben，1 + Lucy + Ben >	< Jesse，1 + Terry + Jesse >	1 + Terry + Jesse >
< Lucy，2 + Lucy + Ben >	< Jone，2 + Jone + Lucy >	< Jone，2 + Jone + Lucy，
< Alice，1 + Jack + Alice >	< Jone，2 + Jone + Jack >	2 + Jone + Jack >
< Jack，2 + Jack + Alice >	< Lucy，1 + Tom + Lucy >	< Lucy，1 + Tom + Lucy，
< Jesse，1 + Jack + Jesse >	< Lucy，1 + Jone + Lucy >	1 + Jone + Lucy，
< Jack，2 + Jack + Jesse >	< Lucy，2 + Lucy + Mary >	2 + Lucy + Mary，
< Alice，1 + Terry + Alice >	< Lucy，2 + Lucy + Ben >	2 + Lucy + Ben >
< Terry，2 + Terry + Alice >	< Mary，1 + Lucy + Mary >	< Mary，1 + Lucy + Mary，
< Jesse，1 + Terry + Jesse >	< Mark，2 + Mark + Terry >	2 + Mark + Terry，
< Terry，2 + Terry + Jesse >	< Mark，2 + Mark + Alma >	2 + Mark + Alma >
< Terry，1 + Philip + Terry >	< Philip，2 + Philip + Terry >	< Philip，2 + Philip + Terry，
< Philip，2 + Philip + Terry >	< Philip，2 + Philip + Alma >	2 + Philip + Alma >
< Alma，1 + Philip + Alma >	< Terry，2 + Terry + Alice >	< Terry，2 + Terry + Alice，
< Philip，2 + Philip + Alma >	< Terry，2 + Terry + Jesse >	2 + Terry + Jesse，
< Terry，1 + Mark + Terry >	< Terry，1 + Philip + Terry >	1 + Philip + Terry，
< Mark，2 + Mark + Terry >	< Terry，1 + Mark + Terry >	1 + Mark + Terry >
< Alma，1 + Mark + Alma >	< Tom，2 + Tom + Lucy >	< Tom，2 + Tom + Lucy，
< Mark，2 + Mark + Alma >	< Tom，2 + Tom + Jack >	2 + Tom + Jack >

（3）Reduce 处理

首先由语句“0 ！ = grandchildnum && 0 ！ = grandparentnum”得知，只要在“value-list”中没有左表或者右表，则不会做处理，可以根据这条规则去除无效的 shuffle 连接。然后根据下面的语句进一步对有效的 shuffle 连接做处理：

```
//左表,取出 child 放入 grandchildren
if('1'= =relationtype){
    grandchild[grandchildnum]=childname;
    grandchildnum+ +;
}

//右表,取出 parent 放入 grandparent
if('2'= =relationtype){
    grandparent[grandparentnum]=parentname;
    grandparentnum+ +;
}
```

针对一条数据进行分析：

```
<Jack, 1+Tom+Jack,
        1+Jone+Jack,
        2+Jack+Alice,
        2+Jack+Jesse >
```

分析结果：左表用“字符 1”表示，右表用“字符 2”表示，上面的 <key, value-list> 中的“key”表示左表与右表的连接键。而“value-list”表示以“key”连接的左表与右表的相关数据。

根据上面针对左表与右表不同的处理规则，取得两个数组的数据见表 8-4。

表 8-4　两个数组的数据

grandchild	Tom、Jone(grandchild[grandchildnum] = childname;)
grandparent	Alice、Jesse(grandparent[grandparentnum] = parentname;)

然后根据下面的语句进行处理：

```
for(int m=0;m<grandchildnum;m+ +){
    for(int n=0; n<grandparentnum;n+ +){
        context. write(new Text(grandchild[m]), new
        Text(grandparent[n]));
    }
}
```

处理结果如图 8-30 所示。

其他的有效 shuffle 连接处理都是如此。最后输出结果和正常关系图相符，实验成功，如图 8-31所示。

Tom	Jesse
Tom	Alice
Jone	Jesse
Jone	Alice

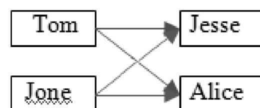


图 8-30　处理结果

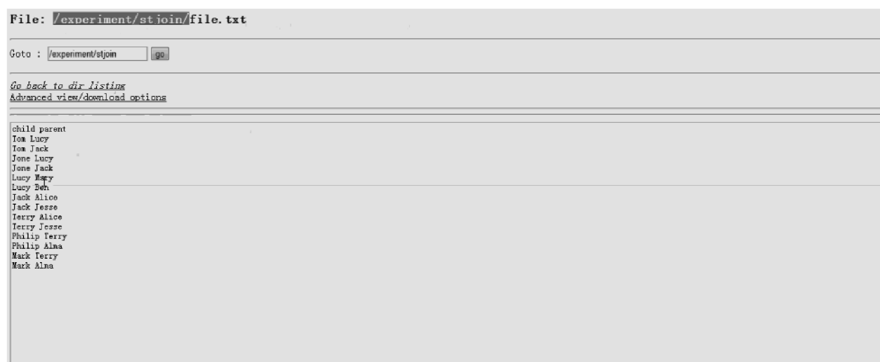


图 8-31　查看输出结果文件

第3篇

项目实训部分

第 9 章　个人存储私有云综合实训

9.1　实验目的

1）了解云存储的原理。
2）掌握云存储的核心技术。

9.2　实验设备

硬件：云计算一体机、PC。

9.3　实验内容

1）操作存储云项目。
2）从后台看到运行过程。

9.4　实验原理

实验主要源码如下：

1. 上传文件

```
public classUploadFileServlet extends HttpServlet {
    private static final longserialVersionUID = 1L;
    private static final Configuration conf = HadoopConfiguration
            . getConfiguration ( );

    publicUploadFileServlet ( ) {
        super ( );
    }
    / *
     * 处理用户提交的文件，把用户提交的文件写入到 HDFS 中
     * * /
    public void doPost (HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //设置 request 编码，主要是为了处理普通输入框中的中文问题
        request. setCharacterEncoding ("utf-8");
```

```
//这里对 request 进行封装，RequestContext 提供了对 request 多个访问方法
RequestContext requestContext = new ServletRequestContext (request);
UserBean ub = (UserBean) request. getSession () . getAttribute ("user");
if (ub = = null) {
    request. getRequestDispatcher ("/login. jsp") . forward (request,
            response);
} else {
    if (FileUpload. isMultipartContent (requestContext)) {
        DiskFileItemFactory factory = new DiskFileItemFactory ();
        //设置文件的缓存路径
        File temp = new File ("/hadoop/tomcat/temp/");
        if (! temp. exists ()) {
            temp. mkdir ();
        }
        factory. setRepository (temp);
        ServletFileUpload upload = new ServletFileUpload (factory);
        //设置上传文件大小的上限，-1 表示无上限
        upload. setSizeMax (1024 * 1024 * 1024);
        List <? > items = new ArrayList ();
        try {
            //上传文件，并解析出所有的表单字段，包括普通字段和文件字段
            items = upload. parseRequest (request);
        } catch (FileUploadException e1) {
            System. out. println ("文件上传发生错误" + e1. getMessage ());
        }
        //下面对每个字段进行处理，分普通字段和文件字段
        Iterator <? > it = items. iterator ();
        while (it. hasNext ()) {
            FileItem fileItem = (FileItem) it. next ();
            //如果是普通字段
            if (fileItem. isFormField ()) {
                System. out. println (fileItem. getFieldName ()
                        + "  "
                        + fileItem. getName ()
                        + "  "
                        + new String (fileItem. getString () . getBytes (
                                "iso8859-1"), "gbk"));
            } else {
                System. out. println (fileItem. getFieldName () + "  "
```

```
                + fileItem. getName ( )  + "  "
                + fileItem. isInMemory ( )  + "    "
                + fileItem. getContentType ( )  + "  "
                + fileItem. getSize ( ) );
//保存文件，其实就是把缓存里的数据写到目标路径下
if (fileItem. getName ( )! = null
        &&fileItem. getSize ( )! =0) {
    //File fullFile = new File (fileItem. getName ( ));
    String [ ] array = fileItem. getName ( ) . split ("\ \ \ \");
    File newFile = new File ("/hadoop/tomcat/"
            + array [array. length-1]);
    try {
        fileItem. write (newFile);
    } catch (Exception e) {
        e. printStackTrace ( );
    }
    String dst = "/tomcat/users/" + ub. getUserId ( )
            + "/files/" + newFile. getName ( );
    InputStream in = new BufferedInputStream (
            new FileInputStream (newFile));
    //开始往 HDFS 中写入文件
    FileSystem fs = FileSystem. get (conf);
    Path path = new Path (dst);
    if (fs. exists (path)) {
        if (newFile. exists ( )) {
            newFile. delete ( );
        }
        request. getRequestDispatcher (
        "/error. jsp? result = 上传的文件已存在! ") . forward (
                request, response);
    } else {
        OutputStream out = fs. create (path,
                new Progressable ( ) {
                    public void progress ( ) {
                        //TODO Auto-generated method
                        //stub
                        System. out. println ("*");
                    }
                });
```

```
                                IOUtils.copyBytes (in, out, 4096, true);
                                IOUtils.closeStream (in);
                                IOUtils.closeStream (out);
                                //写入文件结束
                                FileStatus stat = fs.getFileStatus (path);
                                //更新用户的 sesion 信息
                                ub.setCloudSize (ub.getCloudSize ()
                                        -stat.getLen ());
                                DerbyUtilCase.updateUserStatus (ub);
                                if (newFile.exists ()) {
                                    newFile.delete ();
                                }
                                if (ub.getUserId () .equals ("admin")) {
                                    request.getRequestDispatcher ("/unlimit.jsp")
                                            .forward (request, response);
                                } else {
                                    request.getRequestDispatcher ("/limited.jsp")
                                            .forward (request, response);
                                }
                            }
                        } else {
                            System.out.println ("path is null.");
                        }
                    }
                }
            }
        }
    }
}
```

2. 下载文件

```
public classDownloadFileServlet extends HttpServlet {
    private static final longserialVersionUID = 1L;
    private static final Configuration conf = new Configuration ();

    /*
      *处理用户下载的文件请求，用户提交一个文件名称，系统从 HDFS 读取该文件，然后传输给用户
      */
      public voiddoGet (HttpServletRequest request, HttpServletResponse response)
```

```
        throws ServletException, IOException {
    //找到用户所选定的文件
    request.setCharacterEncoding("utf-8");
    UserBean ub = (UserBean) request.getSession().getAttribute("user");
    if (ub == null) {
        request.getRequestDispatcher("/login.jsp").forward(request,
                response);
    } else {
        //获取用户提交的文件名称
        String uuidname = new String(request.getParameter("filename")
                .getBytes("ISO-8859-1"), "UTF-8");
        File f = new File("/hadoop/tomcat/temp/" + uuidname);
        String dst = "/tomcat/users/" + ub.getUserId() + "/files/"
                + uuidname;
        //开始从 HDFS 上读取文件
        FileSystem fs = FileSystem.get(conf);
        InputStream hadopin = null;
        OutputStream bos = new BufferedOutputStream(new FileOutputStream(f));
        Path hdfsPath = new Path(dst);
        if (! fs.exists(hdfsPath)) {
            request.getRequestDispatcher("/error.jsp? result = 下载资源不存在! ")
                    .forward(request, response);
        } else {
            try {
                hadopin = fs.open(hdfsPath);
                IOUtils.copyBytes(hadopin, bos, 4096, true);
            } finally {
                IOUtils.closeStream(hadopin);
                bos.close();
            }
            //读取文件结束
            String realname = uuidname.substring(uuidname.indexOf("_") + 1);
            //开始给客户端传送文件
            if (f.exists()) {
                //设置应答的相应消息头
                response.setContentType("application/x-msdownload");
                String str = "attachment; filename = "
                        + java.net.URLEncoder.encode(realname, "utf-8");
                response.setHeader("Content-Disposition", str);
```

```
                              //创建一个输入流对象和指定的文件相关联
                              FileInputStream in = new FileInputStream（f）;
                              //从 response 对象中获取到输出流对象
                              OutputStream out = response. getOutputStream（）;
                              //从输入流对象中读数据写入到输出流对象中
                              byte［］ buff = new byte［1024 * 1024］;
                              int len = 0;
                              while（（len = in. read（buff））》0）{
                                   out. write（buff，0，len）;
                              }
                              f. delete（）;
                              in. close（）;
                              out. close（）;
                         } else {
                         request. getRequestDispatcher（"/error. jsp? result = 下载资源不存在！"）
                                   . forward（request，response）;
                         }
                    }
               }
          }
     }
```

9.5　实验步骤

1）首先用浏览器登录云平台 http://master:8080/education。

2）然后输入账户密码登录。管理员账户 admin，密码 admin，其他用户使用前需要注册，如图 9-1 所示。



图 9-1　登录界面

3）登录进去后可以看到五个实验项目，如图 9-2 所示。

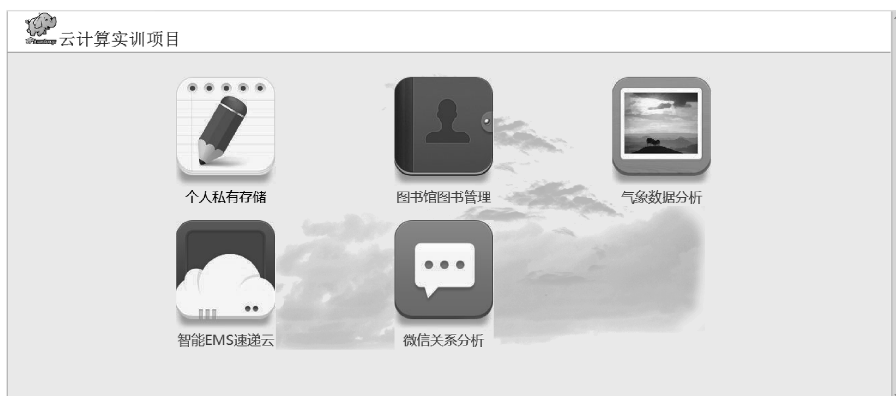


图 9-2　项目界面

4）第一个实验项目：个人私有云存储。点击图标，进入之后会是一个操作界面，如图 9-3 所示。

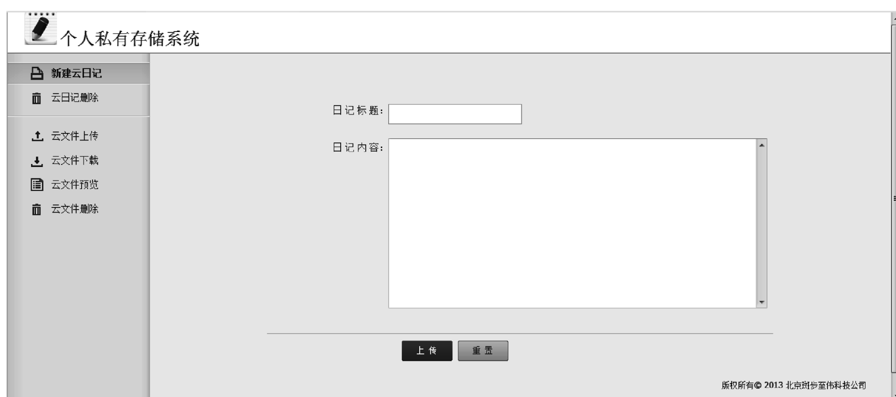


图 9-3　新建云日记

5）首先使用新建日记功能，输入一个标题和日记内容，如图 9-4 所示。然后点击上传，会出现操作成功的页面，如图 9-5 所示。



图 9-4　输入日记内容

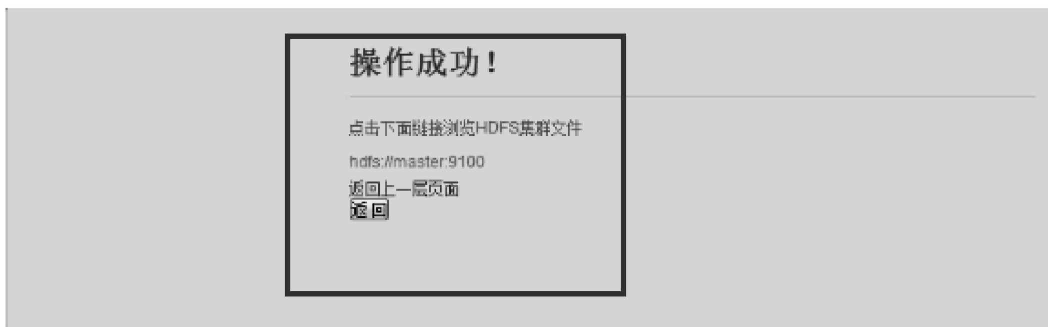


图 9-5　操作成功

6）因为使用的管理员账户登录，所以可以直接点击给出的网址进入到后台查看。

7）跳转到 Hadoop 的文件浏览器，如图 9-6 所示，其他账户是不可以的。

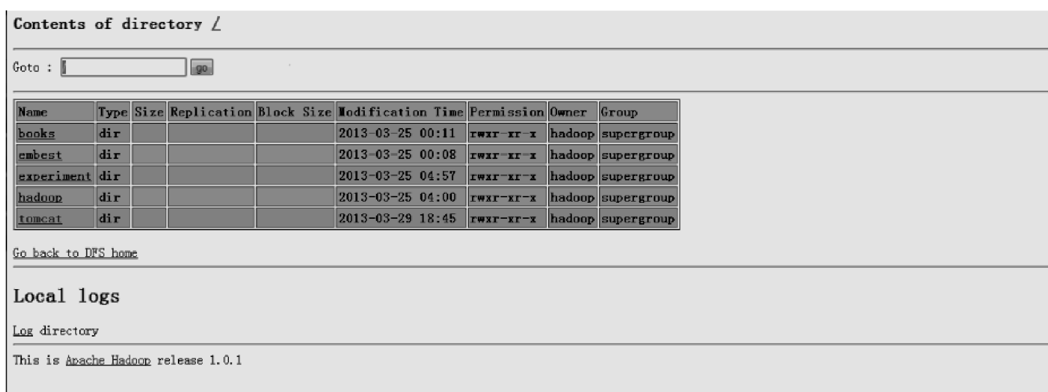


图 9-6　页面浏览文件

8）然后可以通过云文件预览，数据在 notes 里面，如图 9-7 所示。

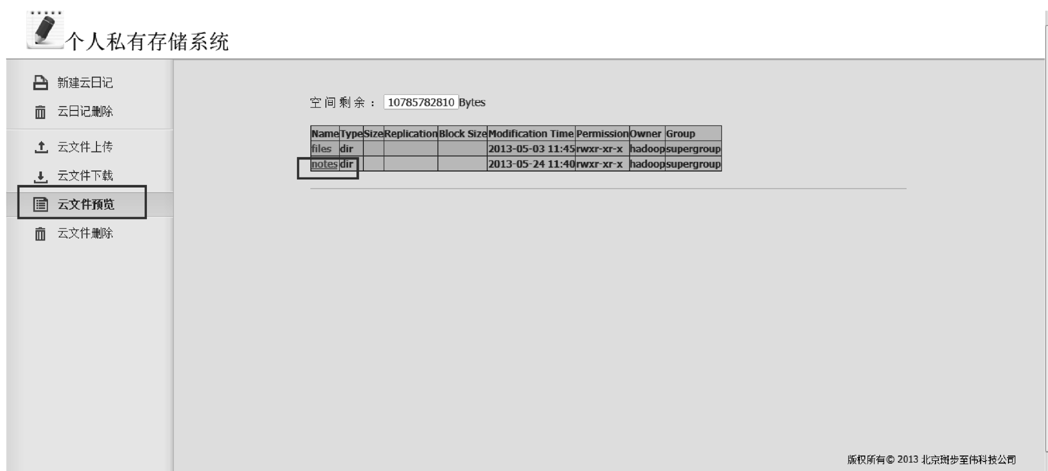


图 9-7　云文件预览文件夹

9）在这里面可以看到已经存在的日记，如图 9-8 所示。

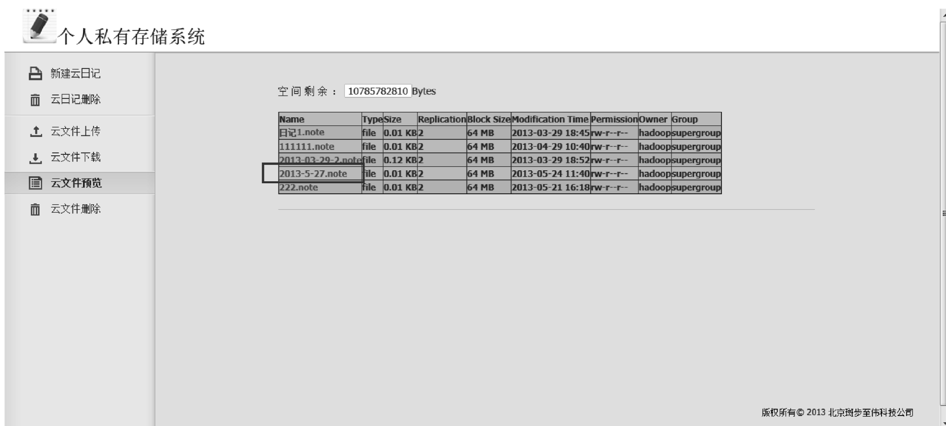


图 9-8　云文件预览文件

10）点击进入可以查看内容，如图 9-9 所示。



图 9-9　云文件浏览内容

11）上传成功之后，存储空间都会相应减少，再上传一个文件，测试一下，如图 9-10 所示。



图 9-10　编写测试日记内容

12）上传后的大小比之前少了，如图 9-11 所示。

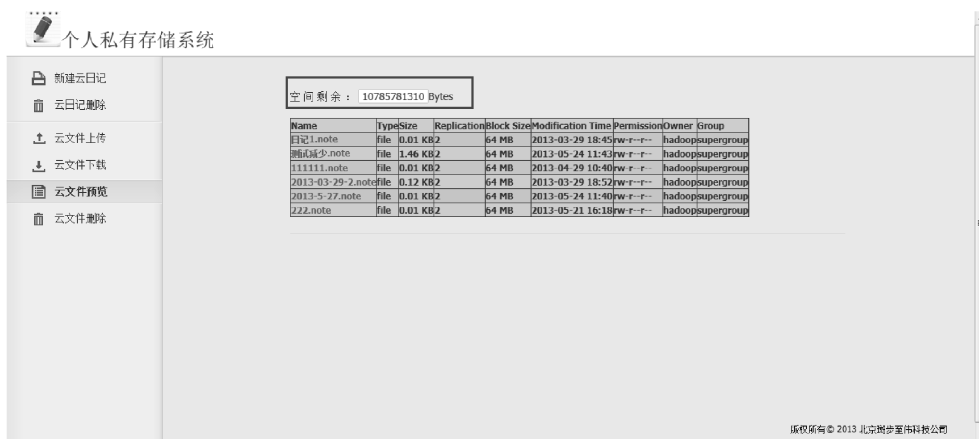


图 9-11　浏览空间剩余

13）之后使用第二个功能，删除刚才测试的文件，如图 9-12 所示。



图 9-12　云日记删除

14）点击删除后会出现操作成功的界面，如图 9-13 所示。



图 9-13　删除成功

15）可以看到目录下已经没有之前的文件了，如图 9-14 所示。

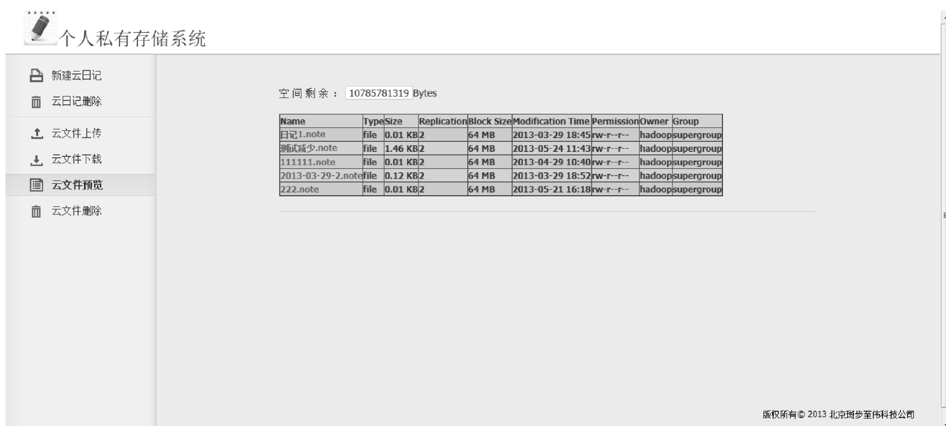


图 9-14　浏览文件夹

16）刚才上传的都是文本文件，还可以上传其他格式的文件，如图 9-15 所示。



图 9-15　云文件上传

17）上传成功后再去浏览一下，如图 9-16 所示。

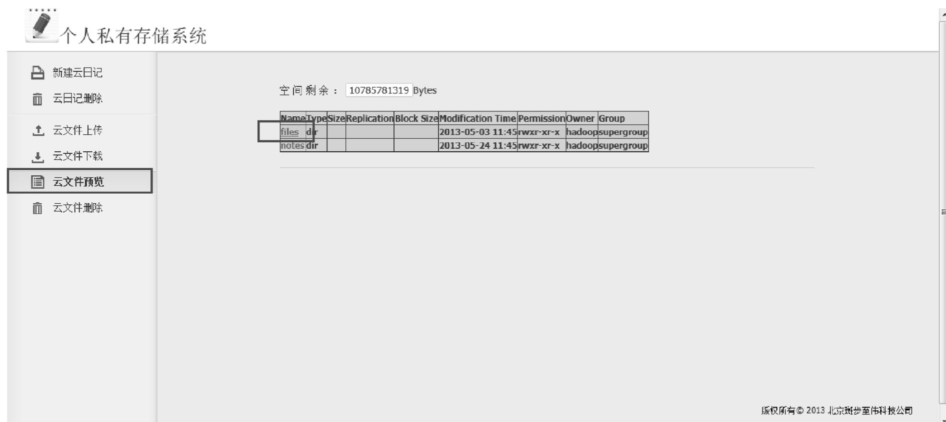


图 9-16　浏览文件夹

18）进入 files 文件夹下，可以看到文件，如图 9-17 所示。



图 9-17　浏览上传文件

19）当看到这个云上有什么资源时，可以通过云文件下载，来获得资源，如图 9-18 所示。



图 9-18　云文件下载

20）点击下载，就可以看到浏览器已经开始下载，如图 9-19 所示。

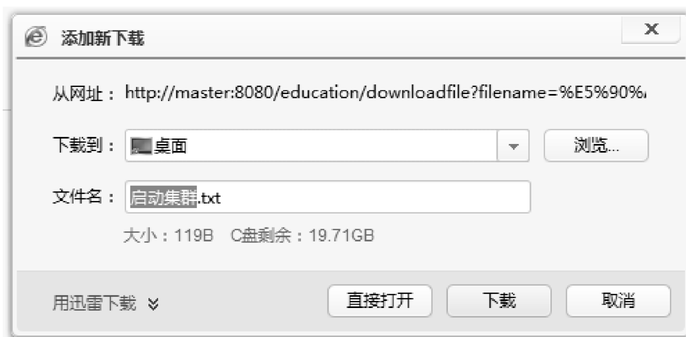


图 9-19　下载界面

21）然后也可以删除云上的文件，如图 9-20 所示。

22）可以看到文件已经删除，如图 9-21 所示。



图 9-20　云文件删除



图 9-21　浏览删除文件

23）上面是云存储最简单的实例，使用到了 HDFS 来制作。

第 10 章　气象数据分析云综合实训

10.1　实验目的

1）熟悉云计算的原理。
2）掌握云计算的根本技术。

10.2　实验设备

硬件：云计算一体机、PC。

10.3　实验内容

1）操作气象数据分析云项目。
2）分析云计算的原理。

10.4　实验原理

实验主要源码如下：

1. 分析天气数据

```
public class WeatherParsingServlet extends HttpServlet{
    private static final long serialVersionUID = 1L;
    private static final Configuration conf = HadoopConfiguration
            .getConfiguration();

    public WeatherParsingServlet(){
        super();
    }

    /*
     * Map 任务，用于从 HDFS 上读取天气数据文件，然后分析这些数据文件，每一条数据代表一日的天气信息，然后把这些数据文件传输给 Reduce 任务
     */
    public static class MeteorologicalMapper extends
            Mapper<LongWritable,Text,Text,MeteorologicalBean>{
```

```
public static enum Counters{
    ROWS
}

//读取一行数据后,该方法就开始处理
public void map(LongWritable key,Text value,Context context)
        throws IOException, InterruptedException{
    String[ ]array = value. toString( ). split("\t") ;
    if(array. length = =2){
        String[ ]metes = array[1]. split(";") ;
        if(metes. length = =3){
            try{
                String[ ]temps = metes[0]. substring(
                        metes[0]. indexOf("(") +1,
                        metes[0]. indexOf(")")). split("/") ;
                MeteorologicalBean mb = new MeteorologicalBean( ) ;
                if(temps. length = =2){
                    mb. setMaxTemp(Float. parseFloat(temps[0]. substring(
                            temps[0]. indexOf("max:") +"max:". length( ),
                            temps[0]. indexOf("℃")))) ;
                    mb. setMinTemp(Float. parseFloat(temps[1]. substring(
                          temps[1]. indexOf("min:") +"min:". length( ),
                            temps[1]. indexOf("℃")))) ;
                }
                String humidity = metes[1]. substring(
                        metes[1]. indexOf("(") +1,
                        metes[1]. indexOf(")")) ;
                mb. setHumidity(Float. parseFloat(humidity. substring(0,
                        humidity. indexOf("%")))) ;
                String WSP = metes[2]. substring(
                        metes[2]. indexOf("(") +1,
                        metes[2]. indexOf(")")) ;
                mb. setWSP(Float. parseFloat(WSP. substring(0,
                        WSP. indexOf("m/s")))) ;
                context. write(
                        new Text(array[0]. substring(0,
                                array[0]. lastIndexOf("-"))),mb) ;
                context. getCounter(Counters. ROWS). increment(1) ;
            }catch(Exception e){
```

```
                    e.printStackTrace();
                }
            }
        }
    }
}

/*
 * Reduce 任务类,该类用户处理 Map 任务提交过来的每日天气数据信息,然后把这些信息相加并平均,最后把结果写入 HDFS 文件当中去
 */
public static class MeteorologicalReducer extends
        Reducer<Text,MeteorologicalBean,Text,Text>{
    public void reduce(Text key,Iterable<MeteorologicalBean>values,
            Context context)throws IOException,InterruptedException{
        int count=0;
        float maxTempTotal=0.0f;
        float minTempTotal=0.0f;
        float humidityTotal=0.0f;
        float WSPTotal=0.0f;
        for (MeteorologicalBean mb:values){
            count++;
            maxTempTotal+=mb.getMaxTemp();
            minTempTotal+=mb.getMinTemp();
            humidityTotal+=mb.getHumidity();
            WSPTotal+=mb.getWSP();
        }
        context.write(key,new Text("AVG{Temp(max:"+maxTempTotal/count
                +"℃/min:"+minTempTotal/count+"℃);Humidity("
                +humidityTotal/count+"%);WSP("+WSPTotal/count
                +"m/s)}"));
    }
}

/*
 * 处理用户提交的天气数据云计算分析,用户提交了分析的命令以后,该方法会向Hadoop 集群提交一个 Map/Reduce 任务,用于执行天气分析的任务
 */
public void doGet(HttpServletRequest request,HttpServletResponse response)
```

```
        throws ServletException,IOException{
    request.setCharacterEncoding("utf-8");
    UserBean ub = (UserBean)request.getSession().getAttribute("user");
    if(ub == null){
         request.getRequestDispatcher("/login.jsp").forward(request,
                   response);
    }else if (ub.getUserId().equals("admin")){
        FileSystem fs = FileSystem.get(conf);
        Path out = new Path("/tomcat/experiment/weathercloud/results");
        if (fs.exists(out)){
            fs.delete(out,true);
        }
        //开始生成处理天气数据的 Map/Reduce 任务 job 信息,然后提交给 Hadoop
集群开始执行
        Job job = new Job(conf,"Parsing Meteorological Data");
        job.setJarByClass(WeatherParsingServlet.class);
        Path in = new Path("/tomcat/experiment/weathercloud/uploaddata");
        FileInputFormat.setInputPaths(job,in);
        FileOutputFormat.setOutputPath(job,out);

        job.setMapperClass(MeteorologicalMapper.class);
        job.setReducerClass(MeteorologicalReducer.class);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(MeteorologicalBean.class);

        job.setNumReduceTasks(1);
        //生成 job 信息完成
        try{
            //提交 job 给 Hadoop 集群,然后 Hadoop 集群开始执行
            job.submit();
            request.getRequestDispatcher("/mrlink.jsp").forward(request,
                    response);
        }catch(Exception e){
            //TODO Auto-generated catch block
            e.printStackTrace();
            request.getRequestDispatcher(
```

```
                            "/error. jsp? result = 任务作业提交失败,请查看集群是否正常
运行 . "). forward(
                            request, response);
                }
            }
        }
    }
```

2. 处理分析结果

```
public class PreviewWeatherResultServlet extends HttpServlet{
    private static final longserialVersionUID = 1L;
    private static final Configuration conf = HadoopConfiguration
            . getConfiguration();

    public PreviewWeatherResultServlet(){
    }

    /*
     * 处理用户提交的浏览云计算天气数据的结果,服务端读取 mapreduce 计算的生成
的结果文件,然后把信息返回给客户端
     */
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException{
        //request. setCharacterEncoding(Charset. defaultCharset(). toString());
        request. setCharacterEncoding("utf-8");
        UserBean ub = (UserBean)request. getSession(). getAttribute("user");
        if(ub = = null){
            request. getRequestDispatcher("/login. jsp"). forward(request,
                    response);
        }else{
            //开始读取结果文件
            FileSystem fs = FileSystem. get(conf);
            Path_success = new Path(
                    "/tomcat/experiment/weathercloud/results/_SUCCESS");
            if(! fs. exists(_success)){
                request. setAttribute("result", null);
                request. getRequestDispatcher("/weatherresult. jsp"). forward(
                        request, response);
            }else{
                request. setAttribute("result","");
```

```
Path reduceRusult = new Path(
        "/tomcat/experiment/weathercloud/results/part-r-00000");
if(fs.exists(reduceRusult)){
    FSDataInputStream fsdis = fs.open(reduceRusult);
    BufferedReader br = new BufferedReader(
            new InputStreamReader(fsdis,"UTF-8"));
    String line = null;
    int count =0;
    float maxTempTotal =0.0f;
    float minTempTotal =0.0f;
    float humidityTotal =0.0f;
    float WSPTotal =0.0f;
    //2012-01
    //AVG{Temp(max:21.754515℃/min:12.678706℃);Humidity
(46.5716%);WSP(19.337095m/s)}
    //开始汇总读取到的每一行数据文件,主要的操作是把所有结果
信息汇总,然后返回给客户端
    while((line = br.readLine())!  =null){
        String[]array = line.split("\t");
        if(array.length = =2){
            MonthBean monBean = new MonthBean();
            String value = array[1].substring(
                    array[1].indexOf("{") +1,
                    array[1].indexOf("}"));
            String[] metes = value.split(";");
            if(metes.length = =3){
                try{
                    String[]temps = metes[0].substring(
                            metes[0].indexOf("(") +1,
                            metes[0].indexOf(")")).split("/
");
                        if(temps.length = =2){

    monBean.setMaxTemp(Float.parseFloat(temps[0].substring(
                                temps[0].indexOf("max:")
                                        +"max:".length(),
                                temps[0].indexOf("℃"))));
                        maxTempTotal + =monBean.getMaxTemp();
```

```
monBean.setMinTemp(Float.parseFloat(temps[1].substring(
                                        temps[1].indexOf("min:")
                                                +"min:".length(),
                                       temps[1].indexOf("℃"))));
                                 minTempTotal += monBean.getMinTemp();
                               }
                               String humidity = metes[1].substring(
                                         metes[1].indexOf("(") +1,
                                         metes[1].indexOf(")"));
                               monBean.setHumidity(Float
                                         .parseFloat(humidity.substring(0,
                                                   humidity.indexOf("%"))));
                               humidityTotal += monBean.getHumidity();
                               String WSP = metes[2].substring(
                                         metes[2].indexOf("(") +1,
                                         metes[2].indexOf(")"));
                               monBean.setWSP(Float.parseFloat(WSP
                                      .substring(0,WSP.indexOf("m/s"))));
                               WSPTotal += monBean.getWSP();
                               count++;
                          }catch(Exception e){
                               e.printStackTrace();
                          }
                     }
                     System.out.println(array[0].split("-")[1]);
                     request.setAttribute(array[0].split("-")[1],
                               monBean);
                }
           }
           //读取结果文件完成
           //2012-12
           //AVG {Temp (max: 21.760002℃/min: 13.001612℃); Humidity
(51.97549%);WSP(21.388714m/s)}
           request.setAttribute("maxTemp",maxTempTotal/count);
           request.setAttribute("minTemp",minTempTotal/count);
           request.setAttribute("humidity",humidityTotal/count);
           request.setAttribute("WSP",WSPTotal/count);
           request.getRequestDispatcher("/weatherresult.jsp").forward(
                     request,response);
```

```
                } else {
                    request.setAttribute("result",null);
                }


            }
        }
    }
}
```

10.5　实验步骤

第三个实验项目：气象数据分析。

1）进入之后是一个操作页面，如图 10-1 所示。



图 10-1　实例文件下载

2）进入之后首先有一个实例下载，点击下载，如图 10-2 所示。



图 10-2　下载界面

3）下载成功，打开查看实例文件内容，如图 10-3 所示。

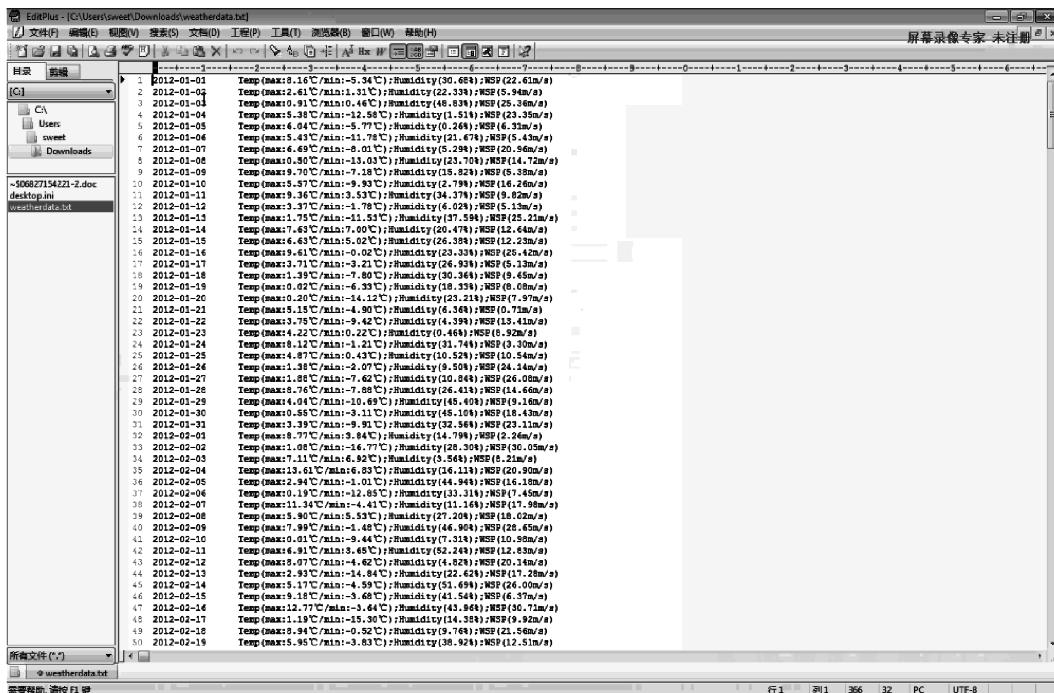


图 10-3　查看实例文件内容

4）这个文件里模拟了一年的天气数据，这里面包括了四个参数，当天的最高、最低气温、湿度和风速，这个实例是提供给学生来修改的，修改文件之后可以上传到云上，如图 10-4 所示。



图 10-4　上传气象数据文件

5）测试时最好一个学生代表一个地区，如图 10-5 所示。

6）上传成功后，到气象数据云计算中看到上传的地区数据，多个学生可以通过自己的账

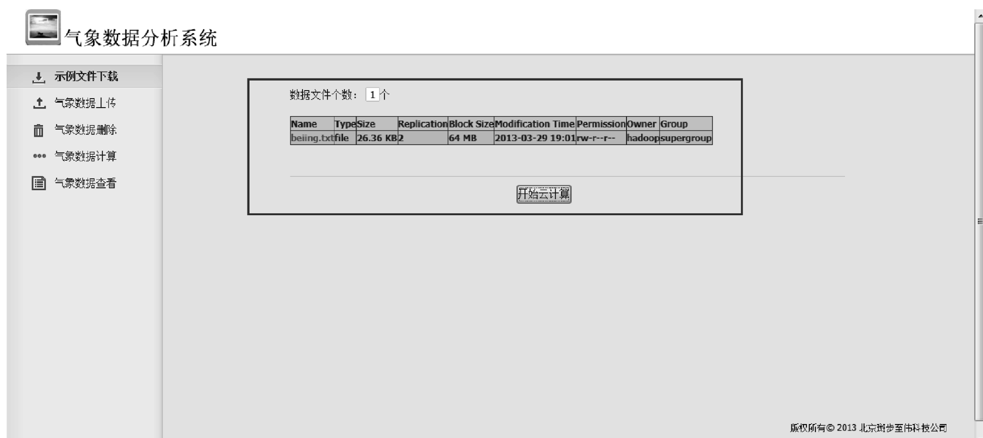


图 10-5　气象数据云计算

户上传文件，然后点击开始云计算，这个界面只有 admin 用户有权限看到，如图 10-6 所示。

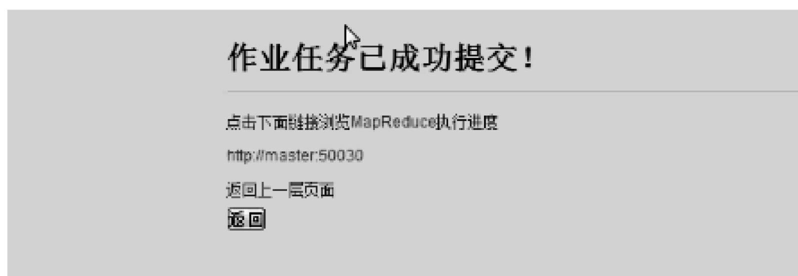


图 10-6　任务提交成功

7）然后可以到后台查看任务，如图 10-7 所示。

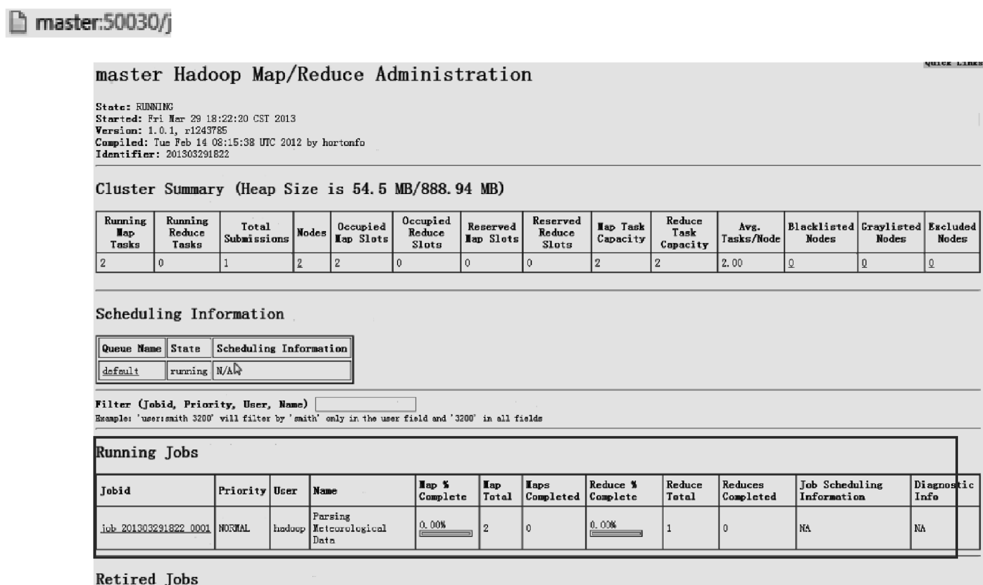


图 10-7　页面浏览任务

8）可以看到有一个任务在运行，可以看到任务的进度，如图 10-8 所示。

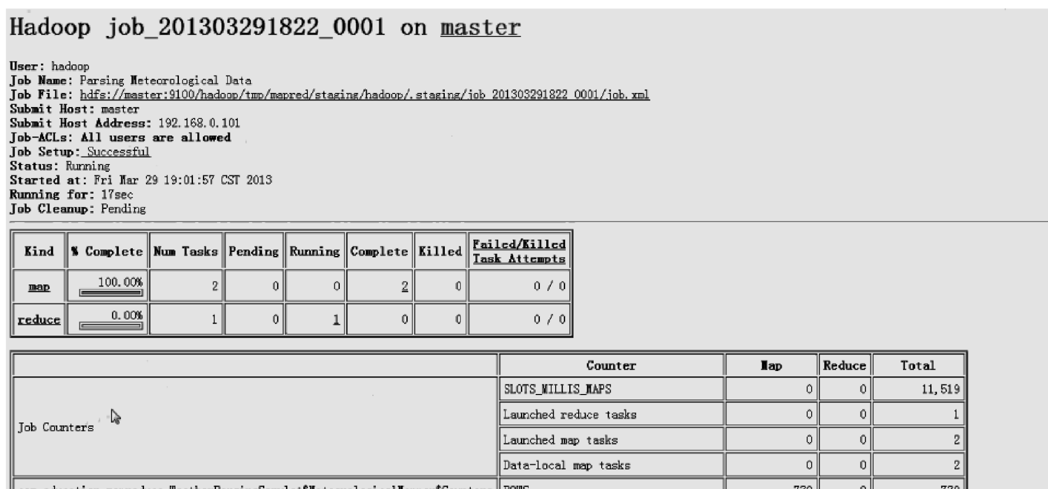


图 10-8　页面浏览任务进度

9）计算完毕后，可以通过气象结果查看，看到全国的平均数据，还有每个月的平均数据，如图 10-9 所示。

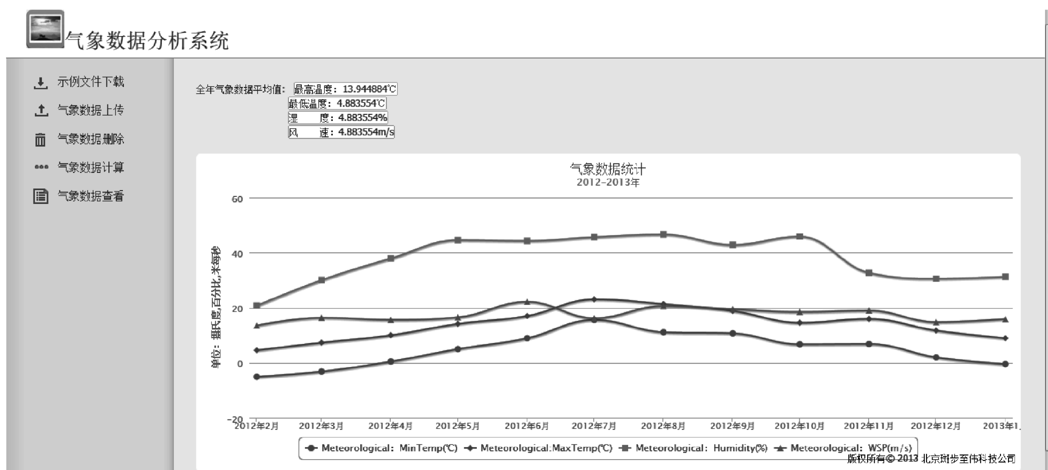


图 10-9　气象数据查看

第 11 章　微信人物关系云综合实训

11.1　实验目的

1）了解人物关系分析的实现原理。
2）能够掌握人物关系分析的技术。

11.2　实验设备

硬件：云计算一体机、PC。

11.3　实验内容

1）操作微信人物关系项目。
2）通过源码更改分析信息。

11.4　实验原理

实验主要源码如下：

```
public class WeixinParsingServlet extends HttpServlet{

    /**
     *
     */
    private static final longserialVersionUID = 1L;
    private static final Configuration conf = HadoopConfiguration
            .getConfiguration();

    public WeixinParsingServlet(){
        super();
    }

    /*
     * Map 任务,主要执行的操作是从数据库中读取微信任务信息,然后从分析条件文件
里读取信息,把这些分析数据准备信息先读取到内存当中
```

```
 *然后开始逐行读取数据文件信息,并开始分析
 */
public static classWeixinMapper extends
        Mapper<LongWritable,Text,Text,Text>{
    private static final SimpleDateFormat sdf = new SimpleDateFormat(
            "yyyy-MM-dd HH:mm:ss");
    private static final byte[] lock = new byte[0];
    private static Map<String,WeixinParsingBean> parsingMap = null;
    private static Map<String,WeixinUserBean> weixinuserMap = null;

    public static enum Counters{
        ROWS
    }

    //每个数据块开始执行之前进行的设置信息,这里主要是从数据库中和分析条件文件中读取预处理的信息,为数据文件的分析做准备
    public void setup(Context context){
        synchronized(lock){
            if(parsingMap == null || weixinuserMap == null){
                parsingMap = new HashMap<String,WeixinParsingBean>();
                weixinuserMap = new HashMap<String,WeixinUserBean>();
                FileSystem fs;
                try{
                    //开始读取分析数据文件
                    fs = FileSystem.get(context.getConfiguration());
                    FileStatus[]uploadparsing = fs
                            .listStatus(new Path(

"/tomcat/experiment/weixincloud/uploadparsing"));
                    for(FileStatus ele:uploadparsing){
                        FSDataInputStream fsdis = fs.open(ele.getPath());
                        String fileName = ele.getPath().getName()
                                .split("\\.")[0];
                        BufferedReader br = new BufferedReader(
                                new InputStreamReader(fsdis,"UTF-8"));
                        String line = null;
                        //timePoint:2013-03-05 13:57:40 durationTime:56
                        //gender:男男
                        //isFriend:否 ageSpan:22 至 22vocation:null
```

//communicationPlace:北京市海淀区,北京市朝阳区 keywords:我爱你

```
try{
    while((line=br.readLine())! =null){
      WeixinParsingBean wpb=new WeixinParsingBean();
        String[]original=line.split("\t");
        if(original.length= =8){
            if(! original[0].contains("null")){
                wpb.setTimePoint(sdf
                        .parse(original[0]
                            .substring(original[0]
                                .indexOf(":")+1))
                        .getTime());
            }

            if(! original[1].split(":")[1]
                    .equals("null")){
                wpb.setDurationTime(Integer
                        .parseInt(original[1]
                                .split(":")[1]));
            }

            String gender=original[2].split(":")[1];
            if(! gender.equals("null")){
                 if(gender! =null){
                     if(gender.equals("男男")){
                         wpb.setGender("11");
                     }else if (gender.equals("女女")){
                        wpb.setGender("00");
                    }else if(gender.equals("异性")){
                        wpb.setGender("01");
                    }else{
                        wpb.setGender("ALL");
                    }
                }
          }

          if(! original[3].split(":")[1]
                    .equals("null")){
```

```
                if(original[3].split(":")[1]
                        .equals("是")){
                    wpb.setFriend("true");
                }else if(original[3].split(":")[1]
                        .equals("否")){
                    wpb.setFriend("false");
                }else{
                    wpb.setFriend("ALL");
                }
            }

            String ageSpan = original[4].split(":")[1];
            if(!ageSpan.equals("null")){
                wpb.setMinAge(Integer
                        .parseInt(ageSpan
                                .split("至")[0]));
                wpb.setMaxAge(Integer
                        .parseInt(ageSpan
                                .split("至")[1]));
            }

            if(!"null".equals(original[5]
                    .split(":")[1])){
                  wpb.setVocations(original[5]
                          .split(":")[1]);
            }
            if(!"null".equals(original[6]
                    .split(":")[1])){
                wpb.setCommunicationPlace(original[6]
                        .split(":")[1].split(","));
            }
            if(!"null".equals(original[7]
                    .split(":")[1])){
                  wpb.setKeywords(original[7]
                          .split(":")[1].split(","));
            }
        }
        parsingMap.put(fileName,wpb);
    }
```

```
                    } catch(Exception e) {
                        e.printStackTrace();
                    }
                    fsdis.close();
                }
                //读取数据库的微信任务信息
                Path path = new Path(
                        "/tomcat/experiment/weixincloud/tmp/DERBY.db");
                FSDataInputStream fsdis = fs.open(path);
                BufferedReader br = new BufferedReader(
                        new InputStreamReader(fsdis,"UTF-8"));
                String line = null;
                while((line = br.readLine())! = null) {
                    String[] array = line.split("\t");
                    try{
                        if(array.length = =6) {
                            WeixinUserBean wub = new WeixinUserBean();
                            wub.setId(Integer.valueOf(array[0]));
                            wub.setName(array[1]);
                            wub.setAge(Integer.valueOf(array[2]));
                            wub.setSex(array[3]);
                            wub.setVocation(array[4]);
                            wub.setFriends(array[5]);
                            weixinuserMap.put(array[0],wub);
                        }
                    } catch(Exception e) {
                        e.printStackTrace();
                    }
                }
            } catch(IOException e) {
                //TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

//数据文件里的每一个行数据都会经过该方法的处理,这里主要分析了每条数据是不是符合用户定义的分析条件,如果符合则把该数据发给 reduce 任务

```
public void map(LongWritable key,Text value,Context context)
        throws IOException,InterruptedException{
    //{10001,10084} \t 2013-02-16 19:22:46 \t 2013-02-16 21:32:45 \t
    //北京市十六中学
    //  \t 办好了?
    String[ ]array = value.toString().split("\t");
    if(array.length = =5){
        try{
        //开始对每行数据进行逐条分析,看是不是满足用户定义的条件信息
            long beginTime = sdf.parse(array[1]).getTime();
            long endTime = sdf.parse(array[2]).getTime();
            String[ ]ids = array[0].substring(
                    array[0].indexOf("{") +1,array[0].indexOf("}"))
                    .split(",");
            if(ids.length = =2){
                for(Map.Entry<String,WeixinParsingBean>me:parsingMap
                        .entrySet()){
                    if(me.getValue().getTimePoint()! =0L){
                        if(beginTime<me.getValue().getTimePoint()){
                            continue;
                        }
                    }
                    if(me.getValue().getDurationTime()! =0){
                        if(endTime-beginTime<1000L*me.getValue()
                                .getDurationTime()*60){
                            continue;
                        }
                    }
                    if(me.getValue().getCommunicationPlace()! =null){
                        boolean pass = false;
                        for(String ele:me.getValue()
                                .getCommunicationPlace()){
                            if(ele.trim().equals(array[3].trim())){
                                pass = true;
                                break;
                            }
                        }
                        if(! pass){
                            continue;
```

```
        }
    }
    if( me. getValue( ). getKeywords( )！ =null) {
        boolean nopass = false;
        for( String ele:me. getValue( ). getKeywords( ) ) {
            if(！ array[4]. contains( ele) ) {
                nopass = true;
                break;
            }
        }
        if( nopass) {
            continue;
        }
    }
    if(！ (me. getValue( ). getGender( ). equals(" ALL") ) ) {
        if( me. getValue( ). getGender( ). equals(" 00 ") ) {
            if(！ ( weixinuserMap. get( ids[0] ). getSex( )
                    . equals("女") ) ) {
                continue;
                }
                if(！ ( weixinuserMap. get( ids[1] ). getSex( )
                        . equals("女") ) ) {
                    continue;
                }
            } else if( me. getValue( ). getGender( )
                    . equals(" 11 ") ) {
                if(！ ( weixinuserMap. get( ids[0] ). getSex( )
                        . equals("男") ) ) {
                    continue;
                }
                if(！ ( weixinuserMap. get( ids[1] ). getSex( )
                        . equals("男") ) ) {
                    continue;
                }
            } else if( me. getValue( ). getGender( )
                    . equals(" 01 ") ) {
                if( weixinuserMap
                    . get( ids[0] )
                    . getSex( )
```

```
                        .equals(weixinuserMap.get(ids[1])
                            .getSex())){
                    continue;
                }
            }
        }
        if(!(me.getValue().getFriend().equals("ALL"))){
            if(me.getValue().getFriend().equals("true")){
                if(!(weixinuserMap.get(ids[0])
                        .getFriends().contains(ids[1]))){
                    continue;
                }
                if(!(weixinuserMap.get(ids[1])
                        .getFriends().contains(ids[0]))){
                    continue;
                }
            }else if(me.getValue().getFriend()
                    .equals("false")){
                if(weixinuserMap.get(ids[0]).getFriends()
                        .contains(ids[1])
                        &&weixinuserMap.get(ids[1])
                            .getFriends()
                            .contains(ids[0])){
                    continue;
                }
            }
        }
        if(me.getValue().getMinAge()>0
                &&me.getValue().getMaxAge()>0){
            if(weixinuserMap.get(ids[0]).getAge()<me
                    .getValue().getMinAge()
                    ||weixinuserMap.get(ids[0]).getAge()>me
                        .getValue().getMaxAge()){
                continue;
            }
            if(weixinuserMap.get(ids[1]).getAge()<me
                    .getValue().getMinAge()
                    ||weixinuserMap.get(ids[1]).getAge()>me
                        .getValue().getMaxAge()){
```

```
                                continue;
                            }
                        }
                        if(me.getValue().getVocations()!=null){
                            if(!(me.getValue().getVocations()
                                    .contains(weixinuserMap.get(ids[0])
                                            .getVocation()))){
                                continue;
                            }
                            if(!(me.getValue().getVocations()
                                    .contains(weixinuserMap.get(ids[1])
                                            .getVocation()))){
                                continue;
                            }
                        }
                        context.write(new Text(me.getKey()),value);
                    }
                }
            }catch(Exception e){
                e.printStackTrace();
            }
            context.getCounter(Counters.ROWS).increment(1);
        }
    }

    public void cleanup(Context context) {
    }
}

/*
 * reduce 任务,主要用来收集 map 任务过程中产生的符合条件的结果信息,然后对这些结果进行用户归并,并把这些结果信息写入到
 * hadoop 的 HDFS 文件系统当中
 */
public static classWeixinReducer extends Reducer<Text,Text,Text,Text>{
    public void reduce(Text key,Iterable<Text> values,Context context)
            throws IOException,InterruptedException{
        String content ="";
        //遍历对结果进行归并
```

```
                for(Text value:values){
                    content + =value.toString() +"\n";
                }
                //context.write(new Text(title),new Text(content));
                //把结果写入到 HDFS 中
                FileSystem hdfs = FileSystem.get(context.getConfiguration());
                Path path = new Path("/tomcat/experiment/weixincloud/results/"
                        + key.toString() +".result");
                if(hdfs.exists(path)){
                    hdfs.delete(path,true);
                }
                FSDataOutputStream hdfsOut = hdfs.create(path);
                hdfsOut.write(content.getBytes());
                hdfsOut.close();
                //写入完成,并关闭相关的资源
            }
        }


        /*
         * 处理用户提交的分析微信数据的云计算请求,客户端提交任务后,服务端负责生成 job 信息,然后把这个 job 信息传递给 Hadoop 集群
         * 让 Hadoop 集群开始执行分析任务
         */
```

11.5　实验步骤

第五个实验项目：微信关系分析。

1）进入之后是操作界面，如图 11-1 所示。

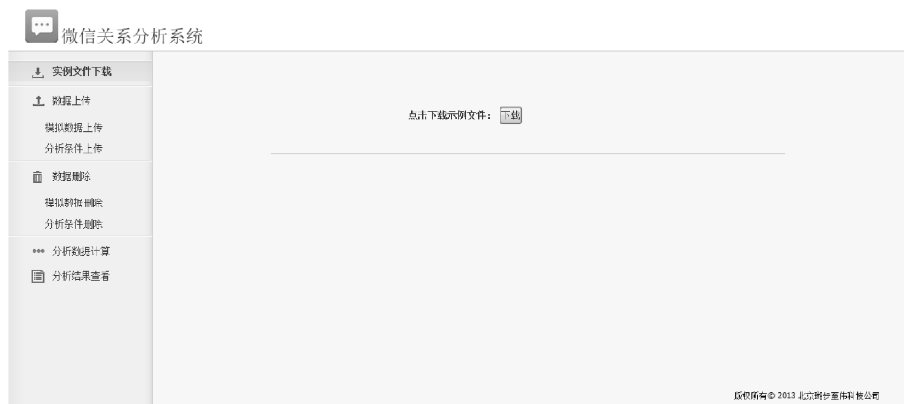


图 11-1　实例文件下载

2）这里还是提供了一个实例下载，如图 11-2 所示。

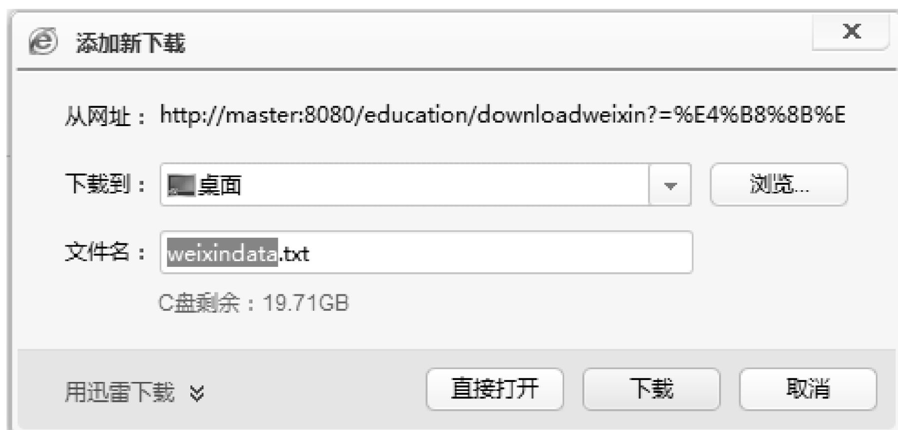


图 11-2　实例下载

3）这个文件是一个数据通信的文件，其中包括一些参数。最前面是微信号，前者和后者联系，然后是从什么时间开始到什么时间结束，在什么地点，内容是什么，如图 11-3 所示。

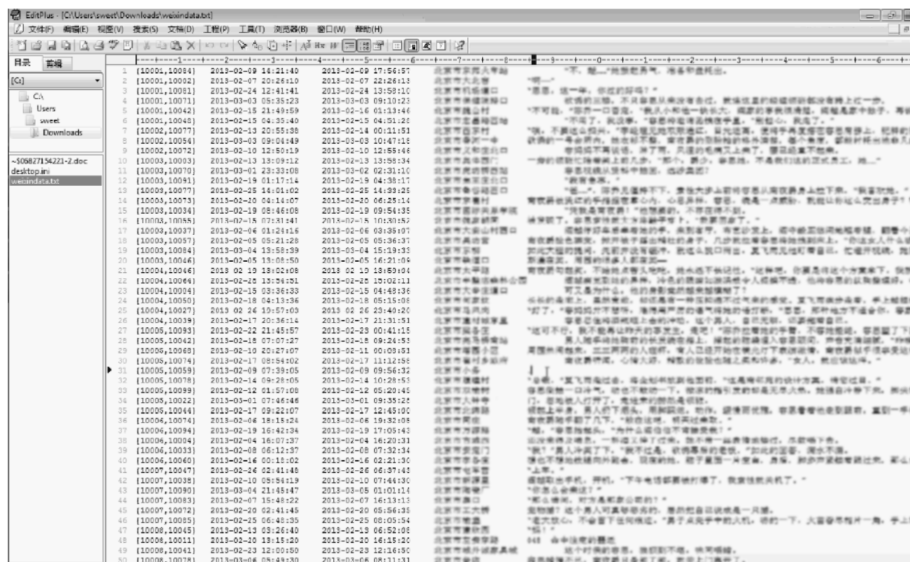


图 11-3　实例文件内容

4）这个是通信数据信息，可以随便更改，不过任务信息是固定的，可以查看一下，如图 11-4 所示。

5）这里模拟了 100 个人物，如图 11-5 所示。

6）人物微信 ID 号为 10001～10100，这个是模拟出来的，所以不可以修改。

7）之后这个实例可以随便添加内容，不过一定要按照格式来。

8）现在记住一条含有《钢铁侠 3》的信息然后上传，如图 11-6 所示。

9）上传之后可以在页面上看到是否上传成功，如图 11-7 所示。

图 11-4 浏览微信用户信息

微信用户信息表预览 返回

用户条数 当前找到 100 条结果

微信ID	姓名	年龄	性别	职业	好友
10001	赵一	31	男	体育	10084,10002,10081,10071,10042,10048
10002	钱二	31	男	建筑	10077,10054,10072
10003	孙三	35	男	体育	10003,10070,10091,10077,10073,10034,10055,10037,10057,10084,10046
10004	李四	24	男	教育	10046,10064,10004,10050,10027,10039
10005	周五	15	女	无业	10093,10042,10068,10074,10059,10078,10099,10022,10044
10006	吴六	17	男	军人	10074,10094,10004,10033,10060
10007	郑七	19	男	律师	10047,10038,10090,10083,10072,10085
10008	王八	30	男	体育	10045,10011,10041,10078,10023,10093
10009	冯九	16	女	教育	10016,10010
10010	陈十	35	女	娱乐	10052,10085,10002,10071
10011	褚一	15	男	体育	10080,10009,10027,10096,10066,10086,10062,10063,10019
10012	卫二	36	女	教育	10005,10061,10057,10062,10007,10079,10033,10087,10056,10019,10082,1006....
10013	蒋三	28	男	餐饮	10010,10010,10095,10080

图 11-5 模拟人物详情

图 11-6 模拟数据上传

图 11-7　页面浏览上传文件

10）现在数据传输上去了，可以开始分析了，现在编辑一下查找的特征，如图 11-8 所示。

图 11-8　分析条件上传

11）上传之后到分析数据云计算中进行计算，如图 11-9 所示。

图 11-9　开始云计算

12）任务提交后，可以在后台看到任务，如图 11-10 所示。

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	2	2	0	0	0	0	2	2	2.00	0	0	0

Queue Name	State	Scheduling Information
default	running	N/A

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201303291822_0002	NORMAL	hadoop	Parsing Weixin Data	0.00%	2	0	0.00%	1	0	NA	NA

图 11-10　浏览执行任务

13）运算完毕后，查看分析结果，如图 11-11 所示。

图 11-11　查看分析结果

第 12 章　云图书馆实例综合实训

12.1　实验目的

1）了解云计算查找的原理。
2）掌握索引和反向索引查找的技术。

12.2　实验设备

硬件：云计算一体机、PC。

12.3　实验内容

1）操作云图书馆项目。
2）通过索引的方式查找更多的图书。

12.4　实验原理

实验主要源码：

1. 上传和索引

```
public class BooksIndexMRThread extends Thread{

    /**
     *该类为一个线程类,每一个用户提交一个课本文件后,系统会为其分配一个线程;该线程执行的主要任务是向 Hadoop 集群提交一个建立索引文件的任务
     *Hadoop 接受到该任务时会开始对文件进行分词,并把分词结果写入到索引文件里
     */
    private static final byte[]lock = new byte[0];
    private static final Configuration conf = HadoopConfiguration
            .getConfiguration();
    private Book book = null;

    public BooksIndexMRThread(Book book){
        this.book = book;
    }
```

```
//线程启动方法,当线程开始执行时,首先启动该方法
public void run() {
	try {
		index(book);
	} catch(IOException e) {
		//TODO Auto-generated catch block
		e.printStackTrace();
	} catch(InterruptedException e) {
		//TODO Auto-generated catch block
		e.printStackTrace();
	} catch(ClassNotFoundException e) {
		//TODO Auto-generated catch block
		e.printStackTrace();
	}
}

public static class BooksMapper extends
		Mapper<LongWritable,Text,NullWritable,NullWritable> {
	private static final Analyzer analyzer = new MaxWordAnalyzer();
	private IndexWriterConfig iwc = null;
	private RAMDirectory ramDir = null;
	private IndexWriter writer = null;
	private StringBuffer sb = null;

	public static enum Counters {
		ROWS
	}
```

//每一个 map 任务执行开始前的准备工作,这里主要是生成了建立索引的对象,并对其进行初始化配置

```
	protected void setup(Context context) {
		sb = new StringBuffer();
		iwc = new IndexWriterConfig(Version.LUCENE_36,analyzer);
		iwc.setOpenMode(OpenMode.CREATE);
		iwc.setRAMBufferSizeMB(128.0);
		iwc.setMaxBufferedDocs(1000);
		ramDir = new RAMDirectory();
		try {
```

```
                writer = new IndexWriter(ramDir,iwc);
            }catch(IOException e){
                //TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
```

//对 map 分配到的任务文件进行读取操作,每读取一个都要进行分词,然后把分词结果写入到内存中,当内存存储到一定程度,刷新到磁盘上

```
        public void map(LongWritable key,Text value,Context context)
                throws IOException,InterruptedException{
            sb.append(value.toString());
            context.getCounter(Counters.ROWS).increment(1);
            if(context.getCounter(Counters.ROWS).getValue()%100 ==0){
                if(sb.toString()! =null&&!"".equals(sb.toString())){
                    Document doc = new Document();
                    doc.add(new Field("name",context.getConfiguration().get(
                            "bookname"),Store.YES,Index.ANALYZED));
                    doc.add(new Field("author",context.getConfiguration().get(
                            "bookauthor"),Store.YES,Index.ANALYZED));
                    doc.add(new Field("publishdate",context.getConfiguration()
                            .get("publishdate"),Store.YES,Index.ANALYZED));
                    doc.add(new Field("sections",sb.toString(),Store.YES,
                            Index.ANALYZED));
                    writer.addDocument(doc);
                    sb = new StringBuffer();
                }
            }
        }
```

//每一个任务执行完成后的清理工作、map 任务完成的索引数据块会以文件夹的目录存在于本地磁盘上,在 map 任务结束时会把这些索引数据块写回到 HDFS 上,最后让 master 来进行最后的合并工作

```
        protected void cleanup(Context context){
            try{
                if(sb.toString()! =null){
                    Document doc = new Document();
                    doc.add(new Field("name",context.getConfiguration().get(
```

```
                    "bookname"),Store.YES,Index.ANALYZED));
                doc.add(new Field("author",context.getConfiguration().get(
                    "bookauthor"),Store.YES,Index.ANALYZED));
                doc.add(new Field("publishdate",context.getConfiguration()
                    .get("publishdate"),Store.YES,Index.ANALYZED));
                doc.add(new Field("sections",sb.toString(),Store.YES,
                    Index.ANALYZED));
                writer.addDocument(doc);
            }
            writer.close();
        }catch(CorruptIndexException e1){
            //TODO Auto-generated catch block
            e1.printStackTrace();
        }catch(IOException e1){
            //TODO Auto-generated catch block
            e1.printStackTrace();
        }

        long timeMillis = System.currentTimeMillis();
        //开始读取本地的磁盘索引文件目录,然后通过文件流的方式把这些文件
回写到 HDFS 上
        String outStr ="/hadoop/booksindex/" + timeMillis;
        File indexDir = new File(outStr);
        IndexWriterConfig fsIndexWriterConfig = new IndexWriterConfig(
                Version.LUCENE_36,analyzer);
        fsIndexWriterConfig.setOpenMode(OpenMode.CREATE_OR_APPEND);
        fsIndexWriterConfig.setRAMBufferSizeMB(128.0);
        fsIndexWriterConfig.setMaxBufferedDocs(1000);
        Directory directory;
        try{
            directory = FSDirectory.open(indexDir);
            IndexWriter fsIndexWriter = new IndexWriter(directory,
                    fsIndexWriterConfig);
            //把内存中的索引库写到文件系统中
            fsIndexWriter.addIndexes(ramDir);
            fsIndexWriter.close();

            FileSystem hdfs = FileSystem.get(context.getConfiguration());
            FileSystem local = FileSystem.getLocal(context
```

```
                    .getConfiguration());
            Path inPath = new Path(outStr);
            String hdfsPath = "/tomcat/experiment/librarycloud/indexes/"
                    + timeMillis + "-lock" + "/";
            FileStatus[] inputFiles = local.listStatus(inPath);
            for(FileStatus ele:inputFiles){
                FSDataOutputStream fsdos = hdfs.create(new Path(hdfsPath
                        + ele.getPath().getName()));
                FSDataInputStream fsdis = local.open(ele.getPath());
                    byte[] buffer = new byte[256];
                    int readByte = 0;
                    while((readByte = fsdis.read(buffer)) > 0){
                        fsdos.write(buffer,0,readByte);
                    }
                    fsdis.close();
                    fsdos.close();
            }
            local.delete(inPath,true);
            hdfs.rename(new Path(hdfsPath),new Path(
                "/tomcat/experiment/librarycloud/indexes/" + timeMillis + "/"));
        }catch(IOException e){
            //TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

//配置 job 相关信息,然后提交给 Hadoop 集群,让其开始执行该 job
private void index(Book book) throwsIOException,InterruptedException,
        ClassNotFoundException{
    conf.set("bookname",book.getName());
    conf.set("bookauthor",book.getAuthor());
    conf.set("publishdate",book.getPublishDate());
    //开始配置 job 信息
    Job job = new Job(conf,"Indexing Book Data");
    job.setJarByClass(BooksIndexMRThread.class);
    String perfix = "/tomcat/experiment/librarycloud/books/"
            + book.getAuthor() + "/" + book.getName();
    Path in = new Path(perfix + ".book");
```

```
Path out = new Path(perfix + "-result");
FileInputFormat.setInputPaths(job,in);
FileOutputFormat.setOutputPath(job,out);

job.setMapperClass(BooksMapper.class);
//job.setReducerClass(null);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(NullOutputFormat.class);
job.setOutputKeyClass(NullWritable.class);
job.setOutputValueClass(NullWritable.class);

job.setNumReduceTasks(0);//没有 reduce 任务
//配置 job 信息完成
//提交 job 到 Hadoop 集群
job.waitForCompletion(true);
//master 服务端开始从 HDFS 读取每一个 map 任务产生的数据块,然后往本地的索引目录下合并
//由于是多线程的操作,在合并的同时可能会是多个线程对本地目录进行操作,以免文件产生错误
//对本地的文件目录加锁,在每一个时刻只允许一个线程对其进行索引文件写入
synchronized(lock){
    FileSystem hdfs = FileSystem.get(conf);
    FileSystem local = FileSystem.getLocal(conf);
    Path indexes = new Path("/tomcat/experiment/librarycloud/indexes");
    FileStatus[] list = hdfs.listStatus(indexes);
    for(FileStatus fs:list){
        if(fs.isDir()&&! fs.getPath().getName().contains("lock")){
            FileStatus[] indexList = hdfs.listStatus(fs.getPath());
            for(FileStatus ele:indexList){
                FSDataInputStream fsdis = hdfs.open(ele.getPath());
                FSDataOutputStream fsdos = local.create(new Path(
                    "/hadoop/indexes/tmp/" + fs.getPath().getName()
                        + "/" + ele.getPath().getName()));
                byte[] buffer = new byte[256];
                int readByte = 0;
            while((readByte = fsdis.read(buffer)) > 0){
                fsdos.write(buffer,0,readByte);
```

```
                    }
                    fsdis.close();
                    fsdos.close();
                }
                hdfs.delete(fs.getPath(),true);
            }
        }
        File indexDir = new File("/hadoop/indexes/tmp");
        Analyzer analyzer = new MaxWordAnalyzer();
        IndexWriterConfig fsIndexWriterConfig = new IndexWriterConfig(
                Version.LUCENE_36,analyzer);
        fsIndexWriterConfig.setOpenMode(OpenMode.CREATE_OR_APPEND);
        fsIndexWriterConfig.setRAMBufferSizeMB(128.0);
        fsIndexWriterConfig.setMaxBufferedDocs(1000);
        Directory directory = FSDirectory.open(new File(
                "/hadoop/indexes/index"));
        File[]arrayFile = indexDir.listFiles();
        for(File file:arrayFile){
            if(file.isDirectory()){
                IndexWriter fsIndexWriter = new IndexWriter(directory,
                    fsIndexWriterConfig);
              //把内存中的索引库写到文件系统中
              Directory tmp = FSDirectory.open(file);
              fsIndexWriter.addIndexes(tmp);
              fsIndexWriter.close();
              tmp.close();
              deleteDir(file);
            }
        }
    }
}

//删除临时的索引文件目录
private void deleteDir(File dir){
    if(dir.isDirectory()){
        File[]children = dir.listFiles();
        for(File ele:children){
            deleteDir(ele);
        }
```

```
            }
            dir.delete();
        }
    }
```

2. 检索文件

```
public class RetrievalBooksServlet extends HttpServlet{
    private static final long serialVersionUID = 1L;
    private static final Analyzer analyzer = newMaxWordAnalyzer();
    private static final File index = new File("/hadoop/indexes/index");

    public RetrievalBooksServlet(){
    }

    /*
     *处理客户端查询课本的请求,服务器端接收到该请求后,读取本地的索引文件,把查询到的结果返回给客户端
     */
    public void doGet(HttpServletRequest request,HttpServletResponse response)
            throws ServletException,IOException{
        //request.setCharacterEncoding(Charset.defaultCharset().toString());
        request.setCharacterEncoding("utf-8");
        UserBean ub = (UserBean)request.getSession().getAttribute("user");
        int pageNumber = 1;
        String pageNumberStr = request.getParameter("pageNumber");
        if(pageNumberStr! = null&&! pageNumberStr.equals("")){
            pageNumber = Integer.parseInt(pageNumberStr);
        }
        if(ub = = null){
            request.getRequestDispatcher("/login.jsp").forward(request,
                    response);
        }else{
            if(! index.exists()){
                request.setAttribute("result",null);
                request.getRequestDispatcher("/error.jsp? result =索引文件不存在!")
                        .forward(request,response);
            }else{
                //获取客户端的查询对象
                Book book = new Book();
                book.setName(new String(request.getParameter("name").getBytes(
```

```
                              "iso-8859-1"),"UTF-8"));
                              book.setAuthor(new String(request.getParameter("author")
                                      .getBytes("iso-8859-1"),"UTF-8"));
                              book.setPublishDate(new String(request.getParameter(
                                      "publishdate").getBytes("iso-8859-1"),"UTF-8"));
                              book.setSection(new String(request.getParameter("section")
                                      .getBytes("iso-8859-1"),"UTF-8"));
                              request.setAttribute("book",book);
                              List<Book> list = null;
                              try{
                                  //根据查询对象在索引文件里进行相关查询,把结果封
装到一个 list 当中,返回给客户端
                              list = retrievalBooks(request,book,pageNumber);
                              request.setAttribute("entryList",list);
                              if(list.size()>0){
                                  request.setAttribute("result","");
                              }else{
                                  request.setAttribute("result",null);
                              }
                              request.getRequestDispatcher("/retrievalbooks.jsp")
                                      .forward(request,response);
                          }catch(Exception e){
                              //TODO Auto-generated catch block
                              e.printStackTrace();
                              request.getRequestDispatcher(
                                      "/error.jsp? result = 检索索引出现异常信息 . ")
.forward(request,response);
                          }
                      }
                      //2012-12
                      //AVG {Temp (max: 21.760002℃/min: 13.001612℃); Humidity
(51.97549%);WSP(21.388714m/s)}
                  }
              }

              //查询索引文件的方法,从索引文件里读取 index 文件,然后根据客户端提交的
条件进行查询,把符合结果的数据返回
              private List<Book> retrievalBooks(HttpServletRequest request,Book quote,
                      int pageNumber)throws Exception{
```

```
List < String > list = new ArrayList < String > ( ) ;
String keywords ="";
if ( quote. getName( ) ! = null&&! "". equals( quote. getName( ) ) ) {
    list. add(" name") ;
    keywords + = quote. getName( ) +",";
}
if( quote. getAuthor( ) ! = null&&! "". equals( quote. getAuthor( ) ) ) {
    list. add(" author") ;
    keywords + = quote. getAuthor( ) +",";
}
if( quote. getPublishDate( ) ! = null
        &&! "". equals( quote. getPublishDate( ) ) ) {
    list. add(" publishdate") ;
    keywords + = quote. getPublishDate( ) +",";
}
if( quote. getSection( ) ! = null&&! "". equals( quote. getSection( ) ) ) {
    list. add(" sections") ;
    keywords + = quote. getSection( ) ;
}
String[ ] fieldArray = ( String[ ] ) list. toArray( new String[ list. size( ) ] ) ;
String[ ] keywordArray = keywords. split(",") ;
BooleanQuery bQuery = new BooleanQuery( ) ;//组合查询
BooleanClause. Occur[ ] flags = new BooleanClause. Occur[ list. size( ) ] ;
for( int index = 0;index < flags. length;index + + ) {
    flags[ index] = BooleanClause. Occur. MUST;
}
Query query = MultiFieldQueryParser. parse( Version. LUCENE_36,
        keywordArray,fieldArray,flags,analyzer) ;
bQuery. add( query,BooleanClause. Occur. MUST) ;

//获取访问索引的接口,进行搜索
Directory directory = FSDirectory. open( index) ;
IndexReader indexReader = IndexReader. open( directory) ;
IndexSearcher indexSearcher = new IndexSearcher( indexReader) ;

int pageSize = 10;
int start = ( pageNumber-1) * pageSize;
int hm = start + pageSize;
request. setAttribute(" pageSize",pageSize) ;
```

```
request. setAttribute(" pageNumber ",pageNumber);

//TopDocs 搜索返回的结果
TopScoreDocCollector res = TopScoreDocCollector. create(hm,false);

indexSearcher. search(bQuery,res);//只返回前 100 条记录
TopDocs tds = res. topDocs(start,pageSize);

int totalCount = res. getTotalHits();
request. setAttribute(" totalPosts ",totalCount);
int pages = (totalCount-1)/pageSize + 1;//计算总页数
request. setAttribute(" totalPages ",pages);
System. out. println("搜索到的结果总数量为:" + totalCount);
System. out. println("总页数为:" + pages);

ScoreDoc[ ] scoreDocs = tds. scoreDocs;//搜索的结果列表
System. out. println(scoreDocs. length);
//创建高亮器,使搜索的关键词突出显示
Formatter formatter = new SimpleHTMLFormatter("<font color = 'red'>",
        "</font>");
Scorer fragmentScore = new QueryScorer(bQuery);
Highlighter highlighter = new Highlighter(formatter,fragmentScore);

Fragmenter fragmenter = new SimpleFragmenter(100);
highlighter. setTextFragmenter(fragmenter);

List<Book> books = new ArrayList<Book>();
//把搜索结果取出放入到集合中,并对关键词进行高亮显示
for(ScoreDoc scoreDoc:scoreDocs){
    int docID = scoreDoc. doc;//当前结果的文档编号
    Document document = indexSearcher. doc(docID);
    Book book = new Book();
    String name = document. get(" name ");
    String highlighterName = highlighter. getBestFragment(analyzer,
            " name ",name);
    if(highlighterName = = null){
        highlighterName = name;
    }
    book. setName(highlighterName);
```

```
                String author = document.get("author");
                String highlighterAuthor = highlighter.getBestFragment(analyzer,
                        "author",author);
                if(highlighterAuthor == null){
                    highlighterAuthor = author;
                }
                book.setAuthor(highlighterAuthor);

                String publishdate = document.get("publishdate");
                String highlighterPublishdate = highlighter.getBestFragment(
                        analyzer,"publishdate",publishdate);
                if(highlighterPublishdate == null){
                    highlighterPublishdate = publishdate;
                }
                book.setPublishDate(highlighterPublishdate);

                String section = document.get("sections");
                String highlighterSections = highlighter.getBestFragment(analyzer,
                        "sections",section);

                if(highlighterSections == null){
                    if(section.length() > 150){
                        highlighterSections = section.substring(0,150);
                    }else{
                        highlighterSections = section;
                    }
                }
                book.setSection(highlighterSections);

                books.add(book);
            }
            indexReader.close();
            indexSearcher.close();
            return books;
        }
    }
```

12.5　实验步骤

第二个实验项目：图书馆图书管理。

1）进入后是一个操作界面，如图 12-1 所示。



图 12-1　书籍上传

2）这个看着很简单，其实后台比较复杂，因为涉及了反向索引技术，这个项目可以让所有的学生上传书籍，然后服务器就会帮助上传的书籍进行索引，这里先简单上传一个文本文件，如图 12-2 所示。



图 12-2　编辑上传书籍信息

3）点击上传文件，如图 12-3 所示。

4）可以在后台看到任务，如图 12-4 所示。

5）上传完毕后进行检索，如图 12-5 所示。



图 12-3　任务提交成功

Cluster Summary (Heap Size is 54.5 MB/888.94 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Graylisted Nodes	Excluded Nodes
0	0	3	2	0	0	0	0	2	2	2.00	0	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201303291822_0003	NORMAL	hadoop	Indexing Book Data	0.00%	1	0	0.00%	0	0	NA	NA

图 12-4　浏览运行任务

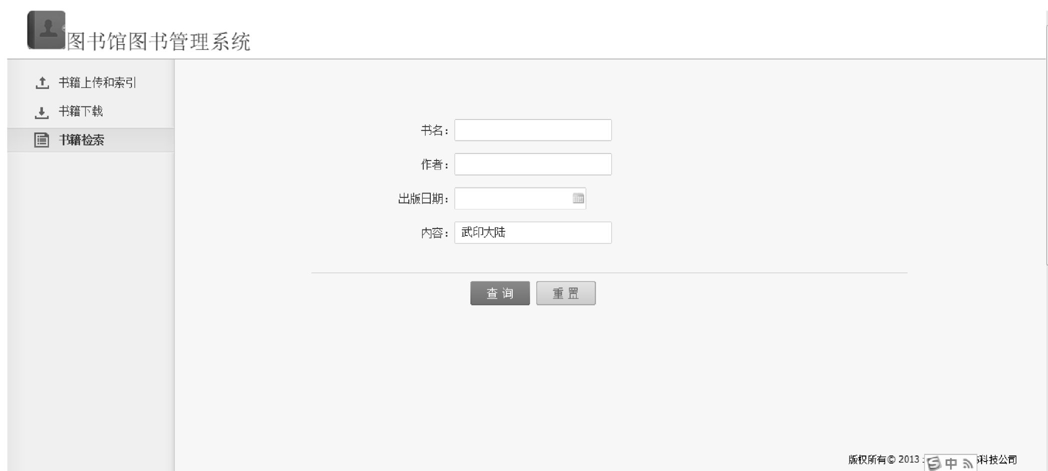


图 12-5　检索书籍

6）点击“查询”按钮就会找到有这个词语的书籍和相关信息，如图 12-6 所示。



图 12-6 查看检索信息结果一

7）多试验几次，这里把一篇文章分成很多小的章节，每 100 行一章，所以当检索作者时会检索出很多，如图 12-7 所示。



图 12-7 查看检索信息结果二

8）这个项目涉及的内容很多，还有分词索引怎么合并之类的内容，大家可以好好分析探讨一下。

第 13 章　物联网与云计算(快递)实例

13.1　实验目的

1）了解物联网和云计算的融合技术。
2）实现用云计算的技术提供更多智慧地球的项目。

13.2　实验设备

硬件：云计算一体机、PC。

13.3　实验内容

1）操作快递云项目。
2）更改源码看是否可以改变功能。

13.4　实验原理

实验主要源码：

```
public class ExpressParsingThread extends Thread{

    /**
     * 此类是一个线程类,当服务器端扫描到新的数据文件后,会启动一个改线程,用于向 Hadoop 集群提交分析数据的 job 任务
     * Hadoop 集群会根据快递数据文件和快递人员的信息来启动 Map/Reduce 任务
     * 分配快递给最近的快递人员,最后把产生的结果信息写入到 HDFS 上
     */
    private static final longserialVersionUID = 1L;
    private static final Configuration conf = HadoopConfiguration
            .getConfiguration();
    private static booleanisLaunch = false;
    private static final Map<String,GPRSBean>map = new HashMap<String,GPRSBean>();
    private static final byte[]lock = new byte[0];
    private static final Map<String,ArrayList<String>>collect = new HashMap<String,ArrayList<String>>();
```

```
    private staticFileSystem fs = null;
    private static Path lockFile = new Path(
            "/tomcat/experiment/expresscloud/results/_lock");

    /**
     * map 任务,负责从 HDFS 上读取快递的数据文件,读取到一行数据,然后遍历所有
的快递人员信息
     * 把此快递分配给离这个快递最近的一个快递人员
     */
    public static classExpressMapper extends
            Mapper<LongWritable,Text,NullWritable,NullWritable>{
        public static enum Counters{
            ROWS
        }

        //启动 map 任务之前的数据准备工作
        protected void setup(Context context){
            try{
                synchronized(lock){
                    fs = FileSystem.get(context.getConfiguration());
                    if(map.size() == 0){
                        Path path = new Path(
                                "/tomcat/experiment/expresscloud/courier/courier.txt");
                        FSDataInputStream fsdis = fs.open(path);
                        BufferedReader br = new BufferedReader(
                                new InputStreamReader(fsdis,"UTF-8"));
                        String line = null;
                        while((line = br.readLine()) != null){
                            String[] array = line.split("\t");
                            try{
                                if(array.length == 2){
                                    GPRSBean gprsb = new GPRSBean();
                                    gprsb.setLongitude(Float
                                    .parseFloat((array[1].split(",")[0])));
                                    gprsb.setLatitude(Float
                                    .parseFloat((array[1].split(",")[1])));
                                    map.put(array[0],gprsb);
                                }
                            }catch(Exception e){
```

```
                        e. printStackTrace( ) ;
                    }
                }
            }
        }
    } catch( Exception e) {
        e. printStackTrace( ) ;
    }
}

//分析快递信息,逐行读取数据文件,一行分析一次
public void map( LongWritable key,Text value,Context context)
        throws IOException,InterruptedException{
    String[ ]array = value. toString( ). split("\t") ;

    if( array. length = =4) {
        try{
            String[ ]ll = array[2]. substring(
                    array[2]. indexOf("经纬度:") +"经纬度:". length( ))
. split(",") ;
            float longitude = Float. parseFloat( ll[0]) ;
            float latitude = Float. parseFloat( ll[1]) ;
            String result = null;
            double distance = -1 ;
            for( Map. Entry < String,GPRSBean > me:map. entrySet( ) ) {
                if( distance = = -1 ) {
                    distance = Math. sqrt( Math. pow( ( longitude-me
                            . getValue( ). getLongitude( ) ) ,2)
                            + Math. pow( latitude
                                    -me. getValue( ). getLatitude( ) ,2) ) ;
                    result = me. getKey( ) ;
                } else if( Math. sqrt( Math. pow( ( longitude-me
                        . getValue( ). getLongitude( ) ) ,2)
                        + Math. pow( latitude
                                -me. getValue( ). getLatitude( ) ,2) ) <
distance) {
                    result = me. getKey( ) ;
                }
            }
```

```
        if(collect.containsKey(result)){
            collect.get(result).add(array[1]+"\t"+array[3]);
        }else{
            collect.put(result,new ArrayList<String>());
            collect.get(result).add(array[1]+"\t"+array[3]);
        }
        context.getCounter(Counters.ROWS).increment(1);
          if(context.getCounter(Counters.ROWS).getValue()%100==0){
            ///tomcat/experiment/expresscloud/results
            synchronized(lock){
                while(fs.exists(lockFile)){
                    Thread.sleep(1000);
                }
                FSDataOutputStream fsdos=fs.create(lockFile);
                fsdos.write("".getBytes());
                fsdos.close();
                for(Map.Entry<String,ArrayList<String>>me:collect
                        .entrySet()){
                    Path user=new Path(
                            "/tomcat/experiment/expresscloud/results/"
                                    +me.getKey()+"/"
                                    +System.currentTimeMillis()
                                    +".order");
                    fsdos=fs.create(user,true);
                    for(String ele:me.getValue()){
                        fsdos.write((ele+"\n").getBytes());
                    }
                    fsdos.close();
                }
                collect.clear();
            }
        }
    }catch(Exception e){
        e.printStackTrace();
        throw new IOException(e);
    }finally{
        if(fs.exists(lockFile)){
            fs.delete(lockFile, false);
        }
```

```
                }
            }
        }

        //清理工作,主要是把此 map 任务产生的结果写入到 HDFS 上,在写入的过程中,会产生同步锁
        protected void cleanup(Context context){
            if(collect.size() > 0){
                synchronized(lock){
                    try{
                        while(fs.exists(lockFile)){
                            Thread.sleep(1000);
                        }
                        FSDataOutputStream fsdos = fs.create(lockFile);
                        fsdos.write("".getBytes());
                        fsdos.close();
                        for(Map.Entry<String,ArrayList<String> > me:collect
                                .entrySet()){
                            Path user = new Path(
                                    "/tomcat/experiment/expresscloud/results/"
                                            + me.getKey() +"/"
                                            + System.currentTimeMillis()
                                            +".order");
                            fsdos = fs.create(user,true);
                            for(String ele:me.getValue()){
                                fsdos.write((ele +"\n").getBytes());
                            }
                            fsdos.close();
                        }
                        collect.clear();
                    }catch(IOExceptione){
                        //TODO Auto-generated catch block
                        e.printStackTrace();
                    }catch(InterruptedException e){
                        //TODO Auto-generated catch block
                        e.printStackTrace();
                    }finally{
                        try{
                            if(fs.exists(lockFile)){
```

```
                        fs.delete(lockFile,false);
                    }
                }catch(IOException e){
                    //TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
        }
    }
}
}

//线程的启动方法,该线程启动时首先调用此方法来启动所有的操作
public void run(){
    isLaunch = true;
    while(isLaunch){
        try{
            FileSystem fs = FileSystem.get(conf);
            Path courier = new Path(
                "/tomcat/experiment/expresscloud/courier/courier.txt");
            if(fs.exists(courier)){
                Path data = new Path(
                    "/tomcat/experiment/expresscloud/uploaddata");
                FileStatus[] dataes = fs.listStatus(data);
                if(dataes.length > 0){
                    Path out = new Path(
                        "/tomcat/experiment/expresscloud/mapreduce");
                    if(fs.exists(out)){
                        fs.delete(out,true);
                    }
                    //初始化 job 的相关信息
                    Job job = new Job(conf,"Parsing Express Data");
                    job.setJarByClass(ExpressParsingThread.class);
                    FileInputFormat.setInputPaths(job,data);
                    FileOutputFormat.setOutputPath(job,out);

                    job.setMapperClass(ExpressMapper.class);
                    //job.setReducerClass(ExpressReducer.class);
```

```
                        job.setInputFormatClass(TextInputFormat.class);
                        job.setOutputFormatClass(NullOutputFormat.class);
                        job.setOutputKeyClass(NullWritable.class);
                        job.setOutputValueClass(NullWritable.class);
                        job.setNumReduceTasks(0);
                        //提交 job 到 Hadoop 集群上,并让其开始执行此任务
                        job.waitForCompletion(true);
                        for(FileStatus ele:dataes){
                            fs.delete(ele.getPath(),true);
                        }
                    }
                }
            }catch(Exception e){
                //TODO Auto-generated catch block
                e.printStackTrace();
            }
            try{
                Thread.sleep(60 * 1000L);
            }catch(InterruptedException e){
                //TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

    public void termination(){
        isLaunch = false;
    }
}
```

13.5　实验步骤

第四个实验项目：智能 EMS 速递云。

1）进入之后是操作界面，如图 13-1 所示。

2）这个项目是在 Hadoop 上又加了一层，点击“启动智能分析”后，就不需要再管了，集群会一直分析数据，这个期间可以随时上传数据进行分析。

3）这里提供了实例下载，如图 13-2 所示。

4）这里面有很多参数，比如快递发出的时间、地址是哪里、经纬度是多少、备注，如图 13-3 所示。

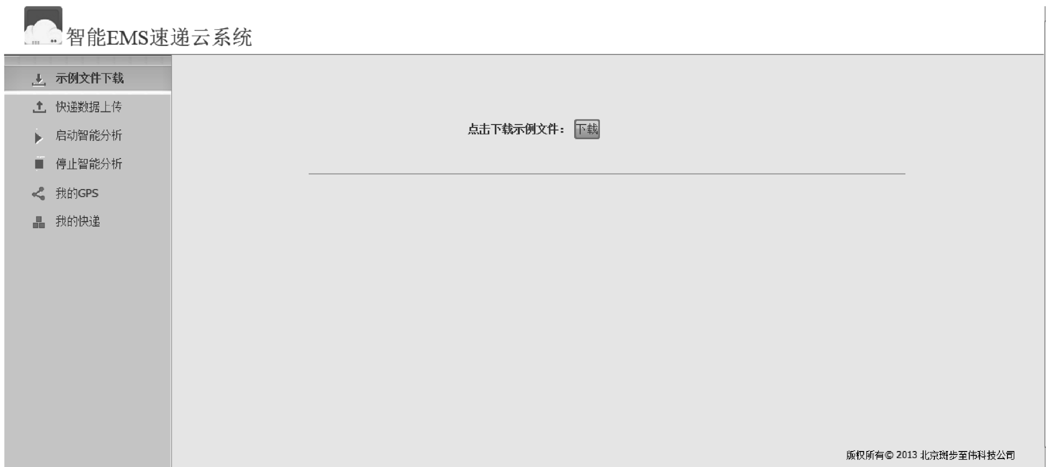


图 13-1　示例文件下载

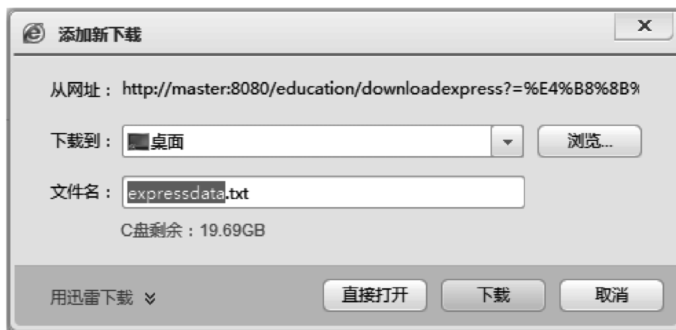


图 13-2　下载界面



图 13-3　查看示例文件内容

5）先更改一条快递进行测试：

地点：北京市东郊火车站经纬度：1.98128，1.90178 备注：如果是周六日，请送到保安处，在三楼 305。

6）首先是启动，如图 13-4 所示。

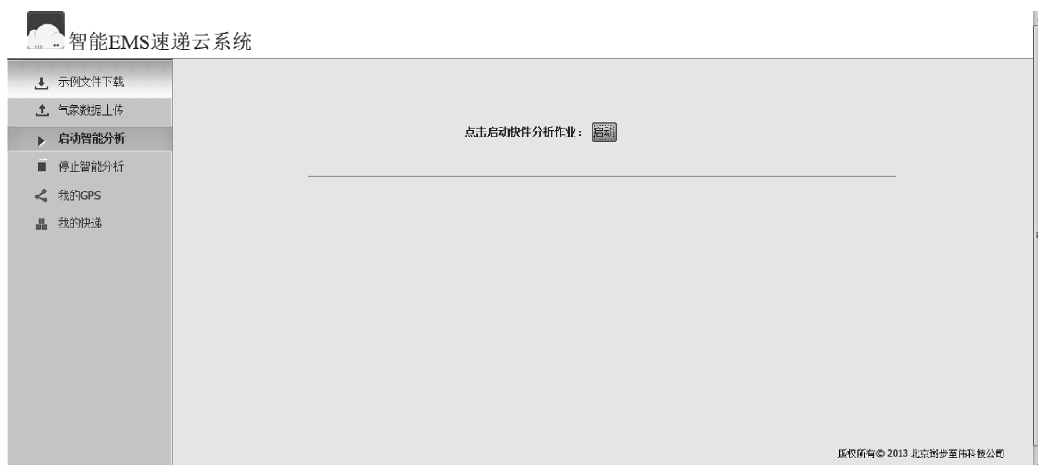


图 13-4　启动云计算

7）用 admin 用户登录，改一下自己的 GPS 位置信息，如图 13-5 所示。



图 13-5　设置自己的 GPS 位置信息

8）更新之后，可以在后台查看到位置信息的更改，如图 13-6 所示。

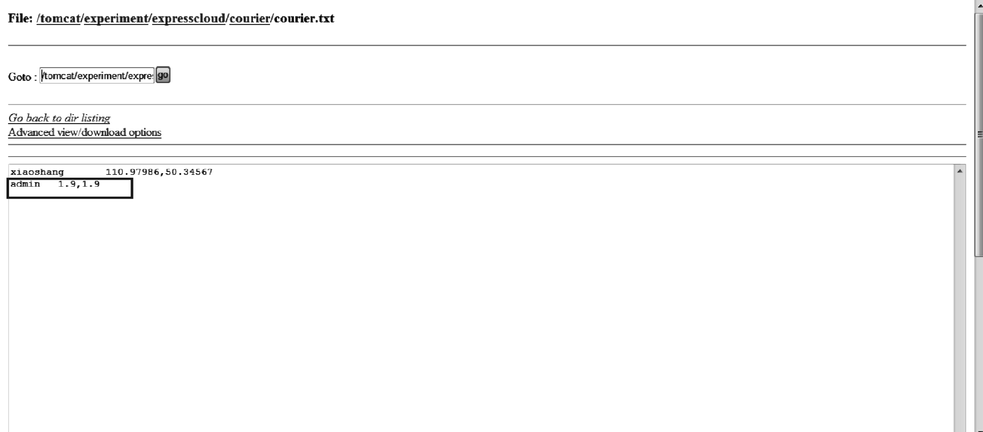


图 13-6　后台查看 GPS 位置信息

9）然后进行快递数据上传，如图 13-7 所示。



图 13-7　快递数据上传

10）由于数据比较多，所以采用文件上传，如图 13-8 所示。



图 13-8　选择上传文件

11）比如自己是一个快递员，如图 13-9 所示。

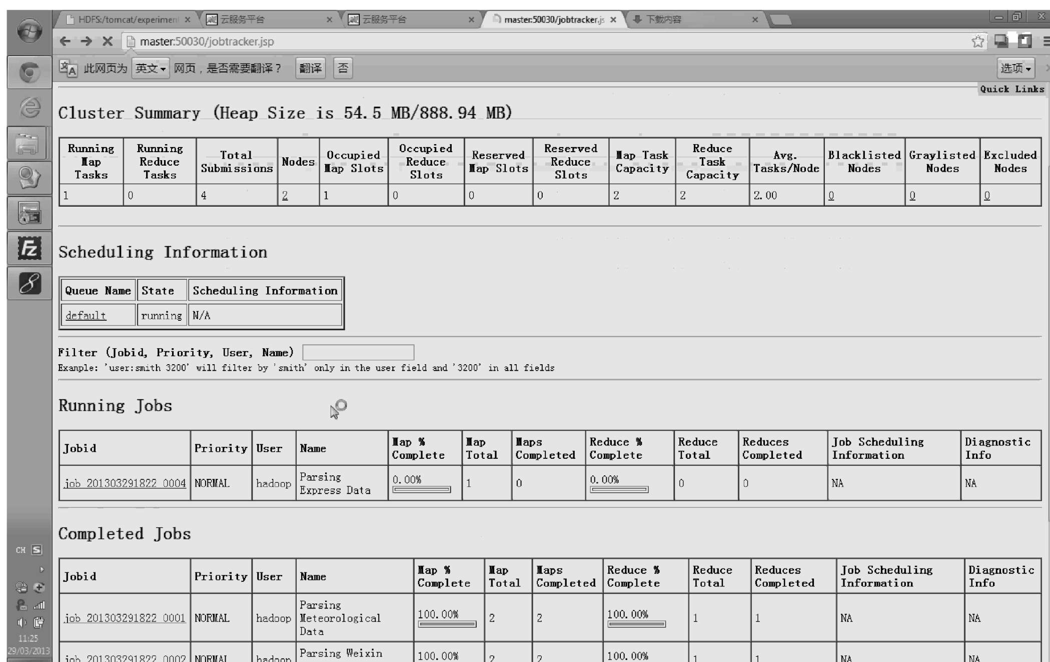


图 13-9　浏览执行任务

12）运行完毕后，查看分给自己的快递信息，搜索一下之前更改位置的快递信息，如图 13-10 所示。



图 13-10　查看分配给自己的快递信息

13）经查看发现这个快递已经发给自己了，当快递员很多时，这样做就会发挥出很大的作用。

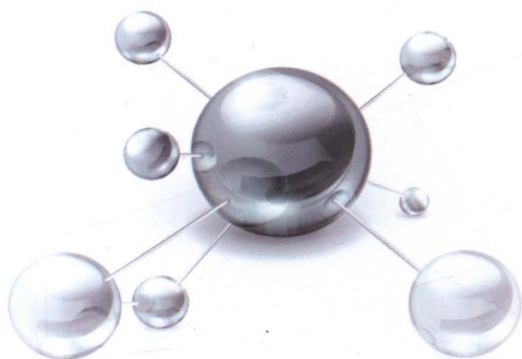
参 考 文 献

[1] 刘鹏. 云计算 [M]. 北京：人民邮电出版社，2011.

[2] Tom White. Hadoop 权威指南(中文版)[M]. 曾大聃，周傲英，译. 北京：清华大学出版社，2010.

[3] Chuck Lam. Hadoop 实战[M]. 韩冀中，译. 北京：人民邮电出版社，2011.

[4] 陆嘉恒. Hadoop 实战[M]. 2 版. 北京：机械工业出版社，2012.



地址：北京市百万庄大街22号
邮政编码：100037
电话服务
社服务中心：010-88361066
销售一部：010-68326294
销售二部：010-88379649
读者购书热线：010-88379203
网络服务
教材网：http://www.cmpedu.com
机工官网：http://www.cmpbook.com
机工官博：http://weibo.com/cmp1952
封面无防伪标均为盗版

上架指导： 工业技术／电子技术／云计算
ISBN 978-7-111-43496-2
策划编辑◎顾　谦/ 封面设计◎赵颖喆



定价：39.90元